Semantic Web And Ontology

Dhana Nandini

Dhana Nandini

Semantic Web And Ontology

Semantic Web And Ontology
1st edition

ISBN 978-87-403-0827-3

Reviewed by Ms. Assistant Professor Swati Ringe

Contents

Learning objective:

The course is aimed at introducing the concepts of Semantic Web. The book covers the basic technologies involved in building the Semantic Web. Students with little prior knowledge on building a web application in web 2.0 (the current web) will easily learn this new technology of Semantic Web. The biggest challenge that all the engineering students may face during their course, is the completion of their final year project. Engineering students who are planning to implement projects in 'Semantic Web' will benefit a lot from this book as it incorporates a practical implementation of Web 3.0.

1 The Revolution Of Web

Objective:

This chapter covers basic concepts related to web, including the history of the web, the different stages of evolution of web etc.

1.1 Basic Terms:

There was a time when scholars and scientists used to travel miles in order to reach the public library which was the only repository of knowledge. The younger generation is smart enough to ask the question "Why don't they Google it?" This is the 21st century we are talking about. The world today revolves around the web. But the dark side of this truth is that very few of us know what exactly is going on in the world of World Wide Web. So, let us try to understand the background of the web.

Following are some of the fundamental terms associated with the web:

World Wide Web:

The World Wide Web (abbreviated as WWW or W3, commonly known as the Web) is a system of interlinked hypertext documents that are accessed via the Internet. With a web browser, one can view web pages that may contain text, images, videos, and other multimedia and navigate between them via hyperlinks.

Hyperlink:

A hyperlink is the most basic building block of the World Wide Web. It is a link from one document, image, word, or web page, to another on the web.

Hypertext:

In 1965, the term 'hypertext' was coined by Ted Nelson to denote a complex, changing and indeterminate file structure. Hypertext is defined as any block of content that includes hyperlinks to other documents, images, or multimedia content.

Hypertext Transfer Protocol:

Hypertext Transfer Protocol, popularly abbreviated as HTTP, is the language that computers use to communicate hypertext documents over the Internet.

Uniform Resource Locator:

Uniform Resource Locator, popularly abbreviated as URL, is the global address of documents and other resources on the World Wide Web.

Hypertext Markup Language:

HTML is the abbreviated form of Hypertext Markup Language. It is a language used to create electronic documents, especially pages on the World Wide Web that contain connections called hyperlinks to other pages.

Every web page you see on the Internet contains HTML code that displays text / images in an easy to read format. Without HTML, a browser will not have any layout for the page and hence would display only plain text without formatting.

Web Browser:

A web browser (commonly referred to as a browser) is a software application for retrieving, presenting and traversing information resources on the World Wide Web.

Web Server:

A web server is any Internet server that responds to HTTP requests to deliver content and services. It can be either hardware or software.

Internet:

The internet is a single worldwide computer network that interconnects other computer networks on which end-user services, such as Web sites or data archives, are located. Thus, it enables the exchange of data and other information.

In response to the launch of Sputnik, the U.S. Defense Department established Advanced Research Projects Agency (ARPA), which eventually focused on computer networking and communication technology. So, ARPANET was originally an experiment conducted to determine how the US military could maintain communication in case of a possible nuclear strike. But later, ARPANET became a civilian experiment that connected university mainframe computers for academic purposes. The original ARPANET grew into the Internet. The internet was based on the idea that there would be multiple independent networks of rather arbitrary design, beginning with the ARPANET as the pioneering packet switching network. Today, the internet has grown into a storehouse of trillions of personal, government, and commercial computers that are connected together by cables and wireless signals.

Web application:

A web application is any application which resides on a server, but is meant to be used by humans. Web applications use web pages as the presentation layer. User interactivity (Graphic User Interface) is done through web pages, but the data is stored and (mostly) manipulated on the server.

Web pages can be static, dynamic or active:

Static web pages –

Static web pages contain the same pre-built content each time the page is loaded. Standard HTML pages are static web pages. They contain HTML code, which defines the structure and content of the web page. Each time an HTML page is loaded, it looks the same. You can find if a page is static or dynamic by looking at the page's file extension in the URL. If it is ".htm" or ".html," the page is probably static.

Dynamic web pages –

Dynamic implies changing or lively. A server-side dynamic web page is a web page whose construction is controlled by an application server, processing server-side scripts. Web pages, such as PHP, ASP and JSP pages are dynamic web pages. These pages contain «server-side» code, which allows the server to generate unique content each time the page is loaded. For example, the server may display the current time and date on the web page. Many dynamic pages use server-side code to access database and generate content from information stored in the database. Websites that generate web pages from database information are often called database-driven websites. If the file extension is ".php", ".asp" or ".jsp" then the page is most likely dynamic.

Active pages –

Active pages are more dynamic than "dynamic" web pages. Internal interaction takes place at the client-side itself and is not dependent on server for interaction.

Web services:

Web Services are server-based applications which may be accessed over the web via HTTP, but is meant primarily for interaction with other programs. Web services are application components or "libraries" which can be used by other applications. Once a web service is deployed, other applications can discover and invoke the deployed service.

Note:

People commonly think that Internet and 'WWW' are the same. But these are two different technologies that are partially related to each other.

The internet is a network of networks that connects millions of computers together globally, forming a network in which any computer can communicate with any other computer as long as they are connected to the Internet. The WWW is a way of accessing information over the medium of the Internet.

1.2 How it all happened

World Wide Web (WWW) was invented by Tim Berners-Lee, a British Computer scientist, in 1990. Prior to the invention of WWW, there came out a queue of technical inventions that eventually led to the invention of WWW.

In 1945, Vannevar Bush wrote in "Atlantic Monthly" about a memory extension called "Memex" which was a photo-electrical-mechanical device that linked documents on microfiche. In 1962, Doug Engelbart devised NLS i.e. an "online System" for browsing and editing information. In the process, he invented the computer mouse. And in 1965, Ted Nelson coined the term hypertext for a complex, changing, indeterminate file structure.

Tim Berners-Lee at CERN in Switzerland wrote software project called ENQUIRE. It was a simple hypertext program that had some of the same ideas as that of web and Semantic Web but was different in several important ways. Combining the work of Vannevar Bush, Ted Nelson and Doug Engelbart, Tim Berners-Lee wrote the Hypertext Transfer Protocol (HTTP). He also implemented a scheme for locating the documents. According to the scheme, every document was assigned a Universal Resource Locator, or URL that served as their address. By the end of 1990 Berners-Lee had written the first browser, or client program, for retrieving and viewing documents known as the www. The immediate two outcomes by Tim Berners-Lee, following the www, was the web server software and HTML. Placing all these components at the right place, in 1991, he made his www browser and web server software available on the Internet. This, in short, is the history of the 'www'.

1.2 Working of a Web server

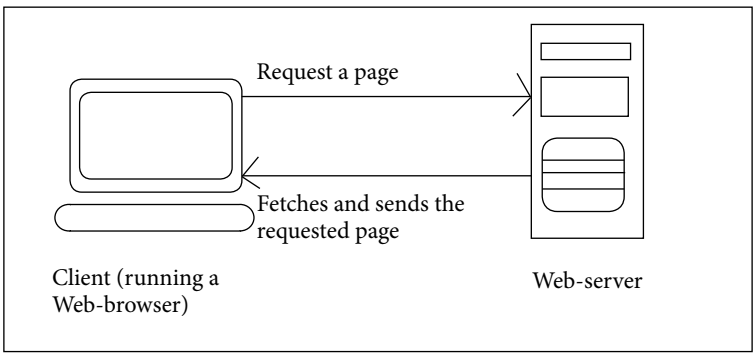


Figure 1.1: Working of a web server

Let's take up a short example in order to understand the working of a web server. Assume that you want to visit Bookboon.com. So you type the URL corresponding to Bookboon.com in the address bar and press Enter key. No matter where in the world the requested page is, it pops up in front of you, on the screen, in fraction of seconds. This activity performed is a basic way to realize the working of a web. Explaining it in a sentence, one can say that the action is initiated by the client machine running a web browser by requesting a page. The server locates the page and sends it back to the client. Thus, responding to a request. Refer to figure 1.1.

1.4 Evolution of Web

Whenever I try to explain Semantic Web to my colleagues the first question that I get is – "Then, what is web 1.0?" Very few of us are aware of the evolutionary hierarchy. The web has gone through tremendous changes before getting the current form. Initially, let us try to understand the evolution of the web.

Web 1.0

The initial form of web was Web 1.0. Web 1.0 was invented by Tim Berners-Lee. It was a read-only platform. Under the Web 1.0 philosophy, companies develop software applications that users can download but they can't see how the application works. For example, Netscape Navigator was a proprietary Web browser of the Web 1.0 era.

Assume an online-dictionary. The purpose of it is to provide us with the meaning of a plethora of words. It has static data. The user can only use it to read but cannot contribute to it. This is the best example of Web 1.0. The drawback of Web 1.0 is that it represented a one-way communication where the users cannot contribute to the web. This forced the world to switch to Web 2.0.

Web 2.0

The traditional Web 1.0 has recently undergone a transformation to become Web 2.0 where the focus is set on folksonomies and collective intelligence. Everything that is famous today in the world of web is Web 2.0. Starting from Facebook till YouTube, everything is Web 2.0. So in short, one can call the current web as Web 2.0. In Web 2.0, the users not only read information from the internet, but also provide information to the web through the internet to share with others. For example, in Facebook you are allowed to write your views, upload photographs and so on. The second generation of the World Wide Web is focused on the ability of people to collaborate and share information online. Web 2.0 is an interactive web. Hence it's called as the Read/Write web.

The characteristics of Web 2.0 are as follows:

- Ability to share views: Web users can contribute to Web 2.0. For example, using an online form, a visitor can add information to Amazon's pages that future visitors will be able to read.
- Using Web pages to link with people: Social networking sites like Facebook and MySpace are popular because they make it easy for users to find each other and keep in touch.
- Fast and efficient ways to share content: YouTube is the perfect example. A YouTube member can create a video and upload it to the site for others to view it.
- New ways to get information: We have countless number of news websites to find information. For example, Wikipedia gives detailed information about almost everything in the world.
- Expanding access to the Internet: Nowadays people access internet not only through computer, but also through mobiles, tablets etc.

Common characteristics of Web 2.0 applications:

- The content is influenced by the user.
- The contents are often generated by the user.
- Applications use the web as a platform.
- Popular trends of the current generation, including Facebook, Twitter, YouTube etc. are leveraged in Web 2.0.
- They include emerging web technologies including Ruby on Rails, RSS, API etc.
- Shareable and editable frameworks in the form of user-oriented to create your own API's.

In current web, the data is presented in such a manner that it is only readable by human and not understandable by machine. So, experts suggested switching over to Web 3.0 in order to make contents machine understandable.

Widgets:

Widgets are small applications which people can insert into web pages by copying and embedding the widget's code into a web page's code. They can be games, news feeds, video players, etc. Some Internet prognosticators believe that Web 3.0 will let users combine widgets together to make mash-ups by just clicking and dragging a couple of icons into a box on a web page. For example, if you want an application that shows you where news stories are happening then just combine news feed icon with a Google Earth icon and Web 3.0 will do the rest.

Web 3.0

Web 3.0 is a Read, Write and Execute web. Web 3.0 is popularly called as the Semantic Web. Some even presume Web 3.0 as a combination of Web 2.0 and Semantic Web. The www has drastically improved access to digitally stored information. However, content in the www has so far only been machine-readable but not machine-understandable. Information in the www is mostly represented in natural language; the available documents are only fully understandable by human beings. The Semantic Web is based on the content-oriented description of digital documents with standardized vocabularies that provide machine understandable semantics. It is the "executable" phase of Word Wide Web with dynamic applications, interactive services, and "machine-to-machine" interaction. Web 3.0 is a Semantic Web which refers to the future. In Web 3.0, computers can interpret information like humans and intelligently generate and distribute useful content tailored to the needs of users.

Web 3.0 can be characterized as follows:

- It has linked data or hyper-data, where data objects are linked to other data objects (similar to how web pages are linked today)
- It has large hyper-data datasets such as DBpedia (a community effort to extract structured information from Wikipedia and make the information available on the Web)
- A query language for hyper-data capable of treating the entire web as a single data center, called SPARQL is needed.
- It is the so-called "Internet of Things" where billions of non-human entities (including houses, cars and appliances) generate and publish their own hyper-data.

If semantics is the study of meaning, think of Web 3.0 as the meaningful web. Very broadly, things on the Internet will be described with descriptor languages, so that computers can understand what they are. Computers will be able to make use of data residing inside web pages. So when you're searching for something, a person, a restaurant, a hotel, the machine goes into its vast network of meaningful linked data, creates connections for you, and suggests useful links that your human mind could never have come up with. At warp speed!

The Web lasagna ends up looking more like this :

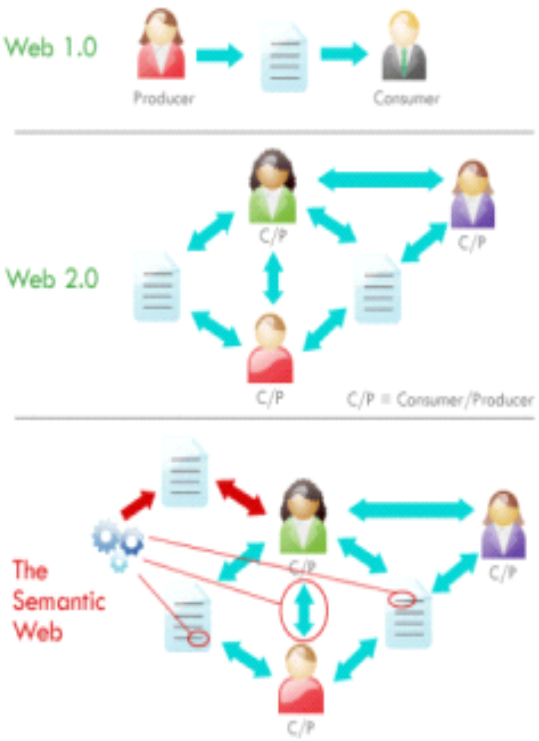


Figure 1.2: Comparing the Webs
Image source: Fredrik Martin

Conclusion:

Have you ever wondered how Google pulls out information? Is the web really intelligent enough to communicate our demands with it? As far as the current scenario is concerned the answer is a big – No. It is this, disability of the web that is being overcome in the next generation web i.e. Web 3.0. Unlike Web 2.0 which focuses on people, Web 3.0 focuses on machines. In short, to explain in just one sentence, web is being made intelligent in order to understand the user demands and serve them better.

2 Need For Semantic Web

Objective:

This chapter covers the following topics:

- Working of the current web.
- Web Crawlers.
- Benefits of Semantic Web.

2.1 Introduction

'Why do we need Semantic Web?' This is another question that would strike anyone who reads about Semantic Web. When people are comfortable with the current web then what's the need to switch to a new platform. The answer to this is explained in the following chapter.

The World Wide Web is not a static container of information, but it's an ever expanding ocean of facts. Every year, on an average, 51 million websites get added to the web. And this figure has meager chances of getting deprecated in the future. It's a proven fact that it will increase in the years about to come. Nowadays, almost all the organizations support open data and make their data available over the web. There was a time when innovation was confined to the four doors of innovation labs. Now is the time when the doors are open to all via open source data. No doubt that more and more information getting added to the web will make it more resourceful, but this will also pose a serious problem too in the near future. We may not know which one is the correct data to be used when we have too much information in front of us.

2.2 Simple Activity

Open Google and type the following two words-

- Joy (press 'Enter' and view the result)
- Delight (press 'Enter' and view the result)

You will get different sets of results in both the cases, though both the words mean the same. This is because the web can't understand that both the words mean the same. At the most basic level, the web pages are considered as strings of words and processed.

Now assume, I am submitting the following two queries to any of the current web's search engines-

Query 1: Domino serves well under heavy load also.

Query 2: Domino serves well under high-demand also.

Query1 is about software called as Domino (An IBM server application platform used for enterprise e-mail, messaging, scheduling and collaboration) that is capable of working well under heavy load as well whereas Query 2 is about a pizza outlet that serves well even under high demand. When submitted for processing, the search engine could not provide an appropriate result for either of the queries. The reason behind this is that the system is not smart. Semantic Web provides this smartness.

2.3 Web 2.0 approach

Semantic Web being a very complex topic, the best way to understand it is to compare and study with the applications and technologies that we are familiar with. So let's study the topic of Semantic Web with the help of our favorite search engine i.e. Google. Before moving on to the algorithm and technic used by Google, let's try to understand some of the basic concepts related to it.

2.3.1 Web Crawler

The main aim of designing a web crawler is to enable the retrieval of web pages and to add their representation to a local repository. A crawler is a program that visits web sites and reads their pages and other information in order to create entries for a search engine index. Web Crawlers are also called as a 'spider'.

Role of a Web Crawler:

- Web Crawlers roam the web with the aim of automating specific tasks related to the web.
- They are responsible for collecting the web-content.

Basic algorithm followed by Web Crawlers:

- Begin with the 'seed' page.
- Create a row/queue for the related pages.
- Retrieve the seed page and process it.
- Extract the URLs they point to.
- Create an entry in the repository.
- Place the extracted URLs in a queue.
- Retrieve each URL from the queue one by one.
- For each of the retrieved URL repeat the above step.

Types of Crawlers:

- **Batch Crawler**
  This type of crawlers crawl a snapshot of their crawl space till they reach a certain size or time limit.

- **Incremental Crawler**
  This type of crawlers keep crawling their crawl space continuously, revisiting the URL so as to ensure freshness.

- **Focused Crawler**
  This type of crawlers, as the name suggests, are quite focused on the pages they crawl. They attempt to crawl pages pertaining to a specific topic and minimize the number of pages that are out of topic and are being collected.

- **Distributed Crawler**
  Distributed web crawling is a distributed computing technique. Many crawlers work together to distribute in the process of web crawling, in order to cover maximum part of the web. A central server manages the communication and synchronization of the nodes, as it is geographically distributed. It basically uses Page rank algorithm for its increased efficiency and quality search. The benefit of a distributed web crawler is that it is robust against system crashes and other events, and can be adapted to various crawling applications.

- **Parallel Crawler**
  Multiple crawlers running in parallel are referred as Parallel Crawlers. A parallel crawler consists of multiple crawling processes called as C-procs which can run on a network of workstations. The Parallel crawlers depend on Page freshness and Page Selection. A Parallel crawler can be on a local network or be distributed at geographically distant locations. Parallelization of crawling system is very vital from the point of view of downloading documents in a reasonable amount of time.

URL Normalization:

Crawlers generally perform some type of URL normalization in order to avoid crawling the same resource again and again. It basically aims at modifying and standardizing a URL in a consistent manner.

Examples of crawlers:

- Scooter
- WebRACE
- RBSE
- Google Crawler
- Www Worm
- Web Fountain

2.3.2 Page Rank

Google uses PageRank to determine the importance of a web page. It is one of the many factors used to determine what all pages must appear in a search result. PageRank measures a web page's importance. Page and Brin's theory is that the most important pages on the Internet are the pages with the maximum number of links leading to them. PageRank considers links as votes, where a page linking to another page is casting a vote. This makes sense, because people do tend to link to relevant content, and pages with more links to them are usually better resources than pages that nobody links. PageRank doesn't stop there. It also looks at the importance of the page that contains the link. Pages with higher PageRank have more weight in "voting" with their links than pages with lower PageRank. It also looks at the number of links on the page casting the "vote." Pages with more links have less weight.

This also makes a certain amount of sense. Pages that are important are probably better authorities in leading web surfers to better sources, and pages that have more links are likely to be less discriminating about where they're linking. PageRank is measured on a scale of one to ten and assigned to individual pages within a website, not the entire website. To find the PageRank of a page, use Google Toolbar. Very few pages have a PageRank of 10, especially as the number of pages on the Internet increases.

2.3.3 How does search engine work?

Step 1:

Crawlers crawl through the web in order to gather the contents about a site that has been changed or a new web-site that has been added, periodically. As mentioned above this work is done periodically and not for each query submitted. The truth is no search engine works in real time.

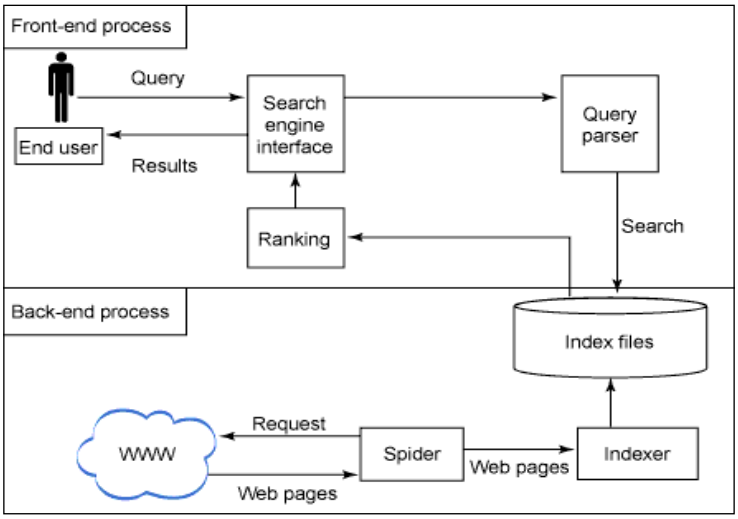


Figure 2.1: Working of Search Engine.

Note:
The crawlers do not cover the entire web. The part of the web that's not covered is called the invisible web.

Step 2:

After gathering the information, the next step is to organize the information. The pages gathered during the crawl process are organized by creating an index, so that we know how to search for it when needed again. The index may include information about words and their locations. When you search, the search terms are searched in the index to find the appropriate pages.

Step 3:

Whenever you submit a query the search engine goes back to its mammoth index library to fetch the required information. Since the search engine finds millions of matching information, it uses an algorithm to decide the order in which the result must be displayed.

2.3.4 How does Google work?

Google runs on a distributed network of thousands of computers and can therefore carry out fast parallel processing. Parallel processing is a method of computation in which many calculations can be performed simultaneously, significantly speeding up data processing. Google works in three parts:

- Googlebot, a web crawler that finds and fetches web pages.
- The indexer that sorts every word on every page and stores the resulting index of words in a huge database.
- The query processor, which compares your search query to the index and recommends the documents that it considers most relevant.

Googlebot

Googlebot is Google's web crawling robot which finds and retrieves pages on the web and hands them off to the Google indexer. It's easy to imagine Googlebot as a little spider scurrying across the strands of cyberspace, but in reality Googlebot doesn't traverse the web at all. It functions much like your web browser, by sending a request to a web server for a web page, downloading the entire page and then handing it off to Google's indexer. Googlebot consists of many computers requesting and fetching pages much more quickly than you can, with your web browser. In fact, Googlebot can request thousands of different pages simultaneously. To avoid overwhelming web servers, Googlebot deliberately makes requests to each individual web server more slowly than it's capable of doing.

When Googlebot fetches a page, it pulls all the links appearing on the page and adds them to a queue for subsequent crawling. Googlebot tends to encounter little spam because most of the web authors link only to what they believe is high-quality pages. By harvesting links from every page it encounters, Googlebot can quickly build a list of links that can cover most part of the web. This technique, known as deep crawling, also allows Googlebot to probe deep within individual sites. Because of their massive scale, deep crawls can reach almost every page in the web. Because the web is vast, this can take some time, so some pages may be crawled only once a month.

Although its function is simple, Googlebot must be programmed to handle several challenges. First, since Googlebot sends out simultaneous requests for thousands of pages, the queue of 'visit soon' URLs must be constantly examined and compared with URLs already in Google's index. Duplicates in the queue must be eliminated to prevent Googlebot from fetching the same page again. Googlebot must determine how often to revisit a page. On one hand, it's a waste of resources to re-index an unchanged page. On the other hand, Google wants to re-index changed pages to deliver up-to-date results.

Google's Indexer

Googlebot gives the indexer the full text of the pages it finds. These pages are stored in Google's index database. This index is sorted alphabetically by search term, with each index entry storing a list of documents in which the term appears and the location within the text where it occurs. This data structure allows rapid access to documents that contain user query terms.

To improve search performance, Google ignores (doesn't index) common words called stop words (such as the, is, on, or, of, how, why, as well as certain single digits and single letters). Stop words are so common that they do little to narrow a search, and therefore they can safely be discarded. The indexer also ignores some punctuation and multiple spaces, as well as converting all letters to lowercase, to improve Google's performance.

Google's Query Processor

The query processor has several parts, including the user interface (search box), the 'engine' that evaluates queries and matches them to relevant documents and the results formatter. PageRank is Google's system for ranking web pages. A page with a higher PageRank is deemed more important and is more likely to be listed above a page with a lower PageRank. Google considers over a hundred factors in computing a PageRank and determining which documents are most relevant to a query, including the popularity of the page, the position and size of the search terms within the page, and the proximity of the search terms to one another on the page. A patent application discusses other factors that Google considers when ranking a page. Visit SEOmoz.org's report for an interpretation of the concepts and the practical applications contained in Google's patent application.

Google also applies machine-learning techniques to improve its performance automatically by learning relationships and associations within the stored data. Google closely guards the formulae it uses to calculate relevance; they're tweaked to improve quality and performance, and to outwit the latest devious techniques used by spammers. Indexing the full text of the web allows Google to go beyond simply matching single search terms. Google gives more priority to pages that have search terms near each other and in the same order as the query. Google can also match multi-word phrases and sentences. Since Google indexes HTML code in addition to the text on the page, users can restrict searches on the basis of where query words appear, e.g., in the title, in the URL, in the body, and in links to the page, options offered by Google's Advanced Search Form and using Search Operators (Advanced Operators).

2.4 Semantic Web's approach

- Knowledge will be organized in conceptual spaces according to its meaning.
- Automated tools will support maintenance by checking for inconsistencies and extracting new knowledge.
- Keyword-based search will be replaced by query answering, i.e. requested knowledge will be retrieved, extracted, and presented in a human friendly way.
- Query answering over several documents will be supported.
- Defining who may view certain parts of information (even parts of documents) will be possible.

In addition to retrieving the results from a search, the way a computer does now (that is to say, systematically, taking one question and pairing it with keywords to get the user millions of answers), a Semantic Web would carry out a more human-like way of solving problems. It would connect not only from A to Z, but also from A to B to C and so on, until it reaches Z.

A Semantic Web reorganizes the vast amount of information that is accessible to us on the internet in a way similar to that of our mind. It would be like training the internet to understand the context surrounding whatever word or phrase being searched through tags the searcher attaches to the subject. The Semantic Web would serve as a connection between human and computer by making the computer think more like a human while still allowing the humans to do the real thinking. This is exhilarating and terrifying at many levels.

For example, let's say that you wanted to have lunch with your friend, Hydie. Then you might have a series of conversation with Hydie. For instance, consider the following set of conversations:

"I have a meeting in my office so cannot go tomorrow afternoon, but after 3 p.m. I am free."

"That will do. Let's go to Meluha?"

"I'm a vegetarian, so that doesn't work for me." (And so on…)

If Semantic Web technology were used in this transaction, Hydie would have an 'agent' that would have access to all kinds of information about her, including her calendar, any food preferences or allergies she might have, and restaurant ratings she's given. Your own agent would have access to similar information about you. These two agents would communicate with one another and then automatically suggest something that makes sense for both of you. They could even make the reservation for you!

More practically, researchers are using Semantic Web technologies to enable machines to infer new facts from existing facts and data. That is, Semantic Web technologies enable computers not only to store and retrieve information, but also to come up with entirely new information on their own.

2.5 Benefits of Semantic Web

- Computers can operate automatically. Since computers can make decisions like people do, they can complete work automatically, thus saving a lot of energy and money.
- Computers can also customize business systems, and companies can run the business more economically requiring less human effort.
- We can use a standardized way to store and query information efficiently.
- Data sharing can be done more easily with the Semantic Web because data warehousing can be distributed. Proper information can help people make instant and correct decisions.
- Facilitates the exchange of content and learning objects.
- Allows learners search learning resources based on semantics, thus making it easier to search their targeted knowledge.
- Improves the context-aware semantic e-learning environments by providing semantic models for context modeling.
- Scalable, reusable, sharable course content.
- The ability to find and move entire course.
- Assemble content to meet the learner's needs.

3 Introduction To Semantic Web

Objective:

This chapter introduces different technologies that are perceived to be the building blocks of the Semantic Web. The chapter focuses on defining the meaning of each and every term and terminology that plays a key role in constructing the Semantic Web.

3.1 Defining Semantic Web

Semantic Web has many well-known definitions. Listed below are a few of them:

Tim Berner Lee's definition for Semantic Web:

"People keep asking what Web 3.0 is. I think maybe when you've got an overlay of scalable vector graphics – everything rippling and folding and looking misty-on Web 2.0 and access to a Semantic Web integrated across a huge space of data, you'll have access to an unbelievable data resource."

Google's CEO, Eric Schmidt stated:

"Web3.0 is a series of combined applications. The core software technology of Web 3.0 is artificial intelligence, which can intelligently learn and understand the semantics. Therefore, the application of Web3.0 technology enables the Internet to be more personalized, accurate and intelligent."

Netflix founder, Reed Hastings thinks that Web 3.0 would be a full video Web as stated below:

"Web 1.0 was dial-up, 50K average bandwidth; Web 2.0 is an average 1 megabit of bandwidth and Web 3.0 will be 10 megabits of bandwidth all the time, which will be the full video Web, and that will feel like Web 3.0."

Yahoo founder Jerry Yang stated at the TechNet Summit in November 2006:

"Web 2.0 is well documented and talked about. The power of the Net reached a critical mass, with capabilities that can be done on a network level. We are also seeing richer devices over last four years and richer ways of interacting with the network, not only in hardware like game consoles and mobile devices, but also in the software layer. You don't have to be a computer scientist to create a program. We are seeing that manifest in Web 2.0 and Web 3.0 will be a great extension of that, a true communal medium…the distinction between professional, semi-professional and consumers will get blurred, creating a network effect of business and applications."

Explanation:

The Semantic Web is an evolution and extension of the existing web that allows computers to manipulate data and information. The idea of the Semantic Web is still undergoing research and development. There is no reliable way to process semantics which questions the purpose of developing such a large-scale project. Yet there is a strong appeal behind the concept of having such an extension to pre-existing documentation of information that is presently distributed on the World Wide Web. There is a great appeal in a web that has the potential ability to 'know' and 'understand' data. This adds a more humanistic quality to standard data processing because the Semantic Web seeks to close the gap between merely providing documents to people and automatic data and information processing. However, in order to reach to this point, developers are challenged by providing a language that can express both data and rules for reasoning.

3.2 Characteristics of Semantic Web

Intelligence:

Experts believe that one of the most promising features of Web 3.0 will be the web with intelligence, i.e., an intelligent web. Applications will work intelligently with the use of Human-Computer interaction and intelligence. Different Artificial Intelligence (AI) based tools & techniques (such as, rough sets, fuzzy sets, neural networks, machine learning, etc.) will be incorporated with the applications to work intelligently. This means, an application based on Web 3.0 can directly do intelligent analysis, and then optimal output would be possible without much user intervention. Documents in different languages can be intelligently translated into other languages in Web 3.0. It should enable us to work through natural language. Therefore, users can use their native language for communication with the others around the world.

Personalization

Another characteristic of Web 3.0 is Personalization. Personal or individual preferences would be considered during different activities such as information processing, searching, forming a personalized portal on the web, etc. Semantic Web would be the core technology for Personalization in Web 3.0.

Reasoning

Semantic Web allows search, integration, answering complex queries, connections and analysis (paths, sub graphs), pattern finding, mining, hypothesis validation, discovery, visualization etc.

Interoperability

Interoperability refers to the aspects such as the seamless integration of data from heterogeneous sources, dynamic composition, interoperation of web services, and the next-generation search engines. Web 3.0 applications would be easy to customize and they can independently work on different kinds of devices. An application based on Web 3.0 would be able to run on many types of Computers, microwave devices, handheld devices, mobiles, TVs, automobiles and many others.

Usability

Usability encompasses new information retrieval paradigms, user interfaces, interaction and visualization techniques, which in turn require methods for dealing with context dependency, personalization, trust and provenance, amongst others, while hiding the underlying computational issues from the user.

Applicability

Applicability refers to the rapidly growing application areas of Semantic Web technologies and methods, the issue of bringing state-of-the-art research results to bear real-world applications, and to the development of new methods and foundations driven by real-application needs, from various domains.

Note 1:

- Semantics with metadata and ontologies for heterogeneous documents and multiple repositories of data, including the web was discussed in 1990s
- Tim Berners-Lee used the term "Semantic Web" in his 1999 book.
- Initial 5 years of Semantic Web research, saw too much of AI/DL, but more practical/applied work has dominated in recent years.

Note 2:

Pervasive web –

Pervasive Web is the term used to describe the phenomenon where the web is operable to a wide range of electronic devices.

3.3 Semantic Web Vs Artificial Intelligence (AI)

In reality, Semantic Web technologies are as much about the data as they are about reasoning and logic. RDF, the foundational technology in the Semantic Web stack, is a flexible graph data model that does not involve logic or reasoning in any way. The realization of the Semantic Web vision does not rely on human-level intelligence. In fact, the challenges are approached in a different way. The full problem of AI is a deep scientific one, perhaps comparable to the central problems of physics (explain the physical world) or biology (explain the living world). In AI, partial solutions may not work.

But on the Semantic Web partial solutions will work. Even if an intelligent agent is not able to come to all the conclusions that a human user might draw, the agent will still contribute to a web much better than the current Web. If the ultimate goal of AI is to build an intelligent agent, exhibiting human-level intelligence (and higher), the goal of the Semantic Web is to assist human users in their day-to-day online activities. It is clear that the Semantic Web will make extensive use of current AI technology and that advances in that technology will lead to a better Semantic Web. But there is no need to wait until AI reaches a higher level of achievement; current AI technology is sufficient to go a long way toward realizing the Semantic Web vision.

3.4 SDLC – An Overview

SDLC stands for software development life cycle. SDLC consists of following activities:

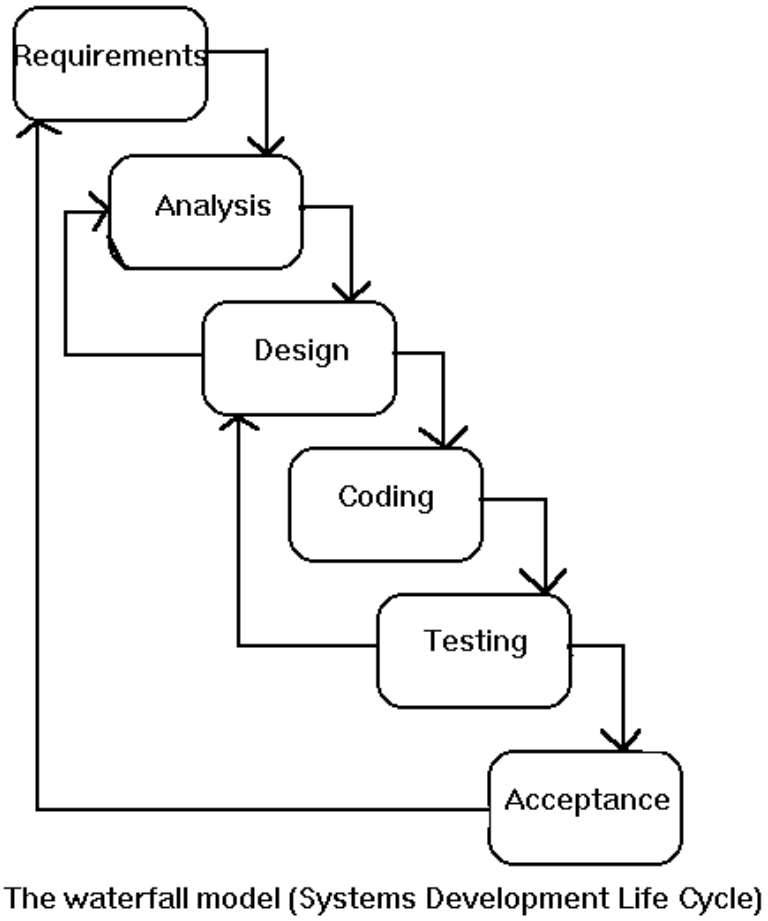


Figure 3.1: SDLC Waterfall model

- **Planning:**
  The most important part of software development, i.e. requirement gathering or requirement analysis is done by the most skilled and experienced software engineers in the organization. After the requirements are gathered from the client, a scope document is created in which the scope of the project is determined and documented.

- **Implementation:**
  Implementation refers to the coding done by the software engineers to implement the client's requirements.

- **Testing:**
  This is a process of finding defects or bugs in the created software. There are different types of testing, performed at different stages of software development. For example, unit testing and regression testing are usually done by developers, whereas smoke-testing and white-box testing are performed by testers.

- **Documentation:**
  Every step in the project is documented for future reference and for the improvement of the software in the development process. The design documentation may include writing the application programming interface (API), the business requirements, intended audience, etc.

- **Deployment and maintenance:**
  At this stage, the software is deployed after it has been approved for release.

- **Maintaining:**
  This is the final stage of the SDLC. Software improvement and new requirements (change requests) can take longer time than the time needed to create the initial development of the software.

3.4.1 There are several software development models followed by various organizations:

Waterfall Model:

This model involves finishing the first phase completely before commencing the next one. When each phase is completed successfully, it is reviewed to see if the project is on track and whether it is feasible to continue.

V-shaped Model:

This model focuses on execution of processes in a sequential manner, similar to the waterfall model, but with more importance placed on testing. Testing procedures are written even before the commencement of coding. A systematic plan is generated before starting the development phase.

Incremental Model:

This life cycle model involves multiple development cycles. The cycles are divided into smaller iterations. These iterations go through a set of phases including requirements, design, implementation and testing. A working version of the software is produced during the first iteration, so working software is created early in the development process.

3.5 Building-blocks of Semantic Web

3.5.1 Ontology

Now, after a deep introduction to Semantic Web, let us try to understand its major building blocks. Let's start with the most important of its kind i.e. Ontology. With ontology, computers can sometimes act as if they 'understand' the information they are carrying. This is where the term "semantic" comes in. In this web, we try to make the meanings so clear that even a computer can understand them. To have truly intelligent systems, knowledge needs to be captured, processed, reused, and communicated. Ontologies support all these tasks. The term 'ontology' can be defined as an explicit specification of conceptualization. The exact meaning depends on the understanding of the terms 'specification' and 'conceptualization'. Explicit specification of conceptualization means that ontology is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents. This definition is consistent with the usage of ontology as a set of concept definitions.

The backbone of ontology is often taxonomy. Taxonomy is a classification of things in a hierarchical form. It is usually a tree or a lattice that expresses sub-assumption relation – i.e., A subsumes B, means that everything that is in A is also in B. An example is classification of living organisms. The taxonomy usually restricts the intended usage of classes, where classes are subsets of the set of all possible individuals in the domain. Ontologies are considered as one of the pillars of the Semantic Web, although they do not have a universally accepted definition. A (Semantic Web) vocabulary can be considered as a special form of (usually light-weight) ontology.

In order to share the knowledge among the agents, an agreement must exist on the topics which are being communicated. This raises the issue of ontological commitment. Ontological commitments allow a number of agents to meaningfully communicate about a domain without necessarily operating on a globally shared theory individually. In the context of multiple agents, a common ontology serves as a knowledge-level specification of the ontological commitments of a set of participating agents. A common ontology defines the vocabulary with which queries and assertions are exchanged among the agents, thereby providing the means to bridge the semantic gap that exists between the lexical representations of information and its non-lexical conceptualization.

3.5.2 RDF/OWL

RDF is a specification that defines a model for representing the world and syntax for serializing and exchanging that model. The W3C has developed an XML serialization for RDF. RDF XML is the standard interchange format for RDF on the Semantic Web, although it is not the only format. For example, Notation3 is an excellent plain text alternative serialization of RDF XML. RDF provides a consistent, standardized way to describe and query Internet resources, from text pages and graphics to audio files and video clips. It offers syntactic interoperability, and provides the base layer for building a Semantic Web. RDF defines a directed graph of relationships. These are represented by object-attribute-value triples, that is, an object O has an attribute A with the value V.

3.5.3 SPARQL

SPARQL is a RDF query language which is capable of retrieving and manipulating the data stored in Resource Description Framework format. It was standardized by the RDF Data Access Working Group (DAWG) of the World Wide Web Consortium, and is recognized as one of the key technologies of the Semantic Web. SPARQL allows a query to consist of triple patterns, conjunctions, disjunctions and optional patterns.

4 Ontology

Objective:

This chapter explains in detail, the role of ontology in building Web 3.0 and the advantage of ontology over traditional hierarchical structures. The chapter also discusses, in short, about protégé which is a tool to generate ontology. The complete procedure to generate ontology using protégé is explained in detail in chapter 6.

4.1 Introduction to Ontology

Going by the traditional definition, 'Ontology can be defined as a branch of metaphysics concerned with the nature and relations of being.' Some questions always get related to ontology, whenever an attempt is made to infer the relation of beings. Stated below are the principal questions involved:

- "What can be said to exist?"
- "Into what categories, if any, can we sort existing things?"
- "What is the meaning of being an entity?"
- "What are the various modes of being an entity?"

We can use the same strategy to build ontology in Semantic Web, i.e. You must make sure that the ontology which you have built answers the above stated fundamental questions. This will help you to conclude that you have included all the essential elements required by your machine to understand the fact that you are trying to put forward. The above mentioned statement's logic will be more meaningful, once you completely understand the concept of ontology.

Definition of Ontology

Ontology is an explicit and abstract modeled representation of already defined finite sets of terms and concepts, involved in knowledge engineering, knowledge management and intelligent information integration. To be more specific, I can define Ontology as an 'explicit specification of conceptualization' (stated by Thomas Gruber).While the terms specification and conceptualization have caused much debate, the essential points of this definition of ontology are:

- Ontology defines (specifies) the concepts, relationships, and other distinctions that are relevant for modeling a domain.
- The specification takes the form of the definitions of representational vocabulary (classes, relations, and so forth), which provide meanings for the vocabulary and formal constraints on its coherent use.

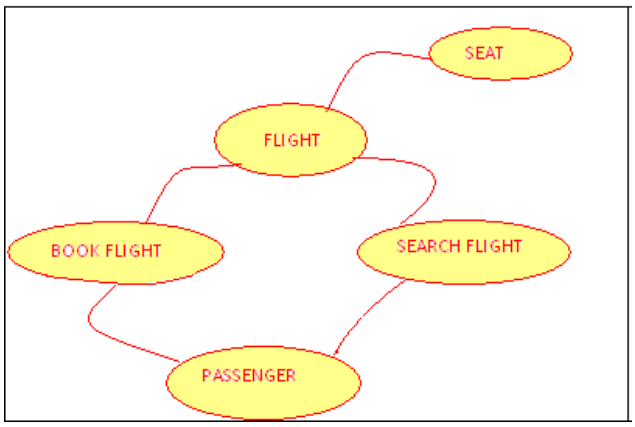


Figure 4.1: Sample Ontology

Conceptualization

Conceptualization contains the objects, concepts and other entities that are assumed to exist in some area of interest and the relationships that hold them. A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose.

A conceptualization can be defined as a tuple (U, R) where,

- U is a set called as a universe of discourse.
- R is a set of relations on U.

A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. Every knowledge base, knowledge-based system, or knowledge-level agent is committed to some conceptualization, explicitly or implicitly.

Note:

In the context of ontology, **formal** means machine understandable. And **share** means consensual knowledge accepted by a group.

4.1.3 Scope of Ontology

Ontology defines a common vocabulary for researchers who need to share information in a domain. It includes machine-interpretable definitions of basic concepts in the domain and the relationship among them. Ontology is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents. In general, the subject of ontology is the study of the categories of things that exist or may exist in some domain. The product of such a study is a catalog of the types of things that are assumed to exist in a domain of interest D from the perspective of a person who uses a language L for the purpose of talking about D. The types in the ontology represent the predicates, word senses, or concept and relation types of the language L when used to discuss topics in the domain D. Logic such as predicate calculus, conceptual graphs, ontology or KIF that is not interpreted, is ontologically neutral. It imposes no constraints on the subject matter or the way the subject may be characterized.

As an interface specification, the ontology provides a language for communicating with the agent. An agent supporting this interface is not required to use the terms of the ontology as an internal encoding of its knowledge. Nonetheless, the definitions and formal constraints of the ontology do put restrictions on what can be meaningfully stated in this language. In essence, committing to ontology (e.g. supporting an interface using the ontology's vocabulary) requires that statements which are asserted on inputs and outputs are logically consistent with the definitions and constraints of the ontology. This is analogous to the requirement that the rows of a database table (or insert statements in SQL) must be consistent with integrity constraints, which are stated declaratively and independently of internal data formats.

Similarly, while ontology must be formulated in some representation language, it is intended to be a semantic level specification, i.e. it is independent of data modeling strategy or implementation. For instance, a conventional database model may represent the identity of individuals using a primary key that assigns a unique identifier to each individual. However, the primary key identifier is an artifact of the modeling process and does not denote something in the domain. Ontologies are typically formulated in languages which are closer in expressive power to logical formalisms such as the predicate calculus. This allows the ontology designer to be able to state semantic constraints without forcing a particular encoding strategy. Similarly, in an ontology one might represent constraints that hold across relations in a simple declaration (A is a subclass of B), which might be encoded as a join on foreign keys in the relational model.

4.2 Switching from database to Ontology

Owing to technological determinism, we are always focused on the next glittering innovation. The one standing in the forefront of this innovation queue is ontology. Being students of computer engineering, developing the front-end and back-end of a web application is no more a big deal, but a challenge is certainly faced when it comes to making the system more intelligent. Under such circumstances, the best way would be to migrate from database to ontology. Ever since the introduction of web 1.0 which is actually a read-only platform for information, to the Web 2.0 which is supposed to be a platform for participation, emphasis has always been laid on developing a more nuanced way to organize information. Basically, ontologies work to organize information. No matter what the domain or scope is, ontology is a description of a world view using a linked or networked graph structure. Taking a little bit of diversion, let's have a look into a more common term, i.e. 'Relational database management system (RDBMS)'. We have many systems that are based on RDBMS. In fact, most of the current folksonomy use RDBMS as their base. The most obvious reason behind this is that database management system is a standard way to store data on a permanent basis and that the extraction of data can be easily done using SQL.

But consider a case wherein you have several databases represented in various formats and your application insists on an integration of these databases, you will face several problems because of their different formats. This is the area where ontology gains weightage. By virtue of the relationship structure underlying ontology, they are excellent vehicles for discovery and linkages. Parsing through this relationship graph is the basis of the Concept Explorer. Separating domain knowledge from operational knowledge and enabling their reuse, sharing a common understanding of the structure of information among software agents are some of the important goals implemented through the medium of ontology. Like for instance, if there are several different web sites containing information about medicines and if these Web sites share and publish the same underlying ontology of the terms they all use, then computer agents can extract and aggregate information from these different sites. The agents can use this aggregated information to answer user queries or as input data to other applications. Thus, they provide knowledge sharing and reuse among both human and computer agents because of their ability to interweave human and machine understanding through formal and real-world semantics.

4.2.1 Taxonomy as a pre-cursor of Ontology

Learning a concept in Semantic Web is not an easy task. This is because, most of the topics are research-oriented and in order to properly understand the definitions of these concepts, an in depth understanding of the terms that construct the definition is a must. The easiest way to do this is by trying to relate these technical terms with their corresponding dictionary meaning. The word taxonomy is derived from two Greek words- Taxis and Nomos which mean 'the arrangement and ordering of things' and 'anything assigned, usage or custom, law or ordinance' respectively. In a formal way, taxonomy can be defined as a subject-based classification that arranges the term in a controlled vocabulary and allows related terms to be grouped together and categorized in ways that make it easier to find the correct term to use.

Many taxonomies have a hierarchical structure, but this is not a requirement. Taxonomies can be explained in simple terms as a graphical representation of classification of things, ideas, etc. According to some taxonomic scheme almost anything can be classified, as long as they have a logical hierarchy. They work towards organizing information. The backbone of ontology is often taxonomy. Taxonomy is a classification of things in a hierarchical form. It is usually a tree or a lattice that expresses the subsumption relation. An example is classification of living organisms. The taxonomy usually restricts the intended usage of classes, where classes are subsets of the set of all possible individuals in the domain.

Example of taxonomy:

When we try to compare the concept of taxonomy and ontology we will find that both of them run on similar concepts. For layman or end-users it's almost the same concept. But technocrats are obliged to perceive them separately. This is because ontology implies a broader perspective of taxonomy. Taxonomy is a mere classification of objects for better understanding of objects and their sub-types, whereas ontology represents a complex network of interconnections between the nodes that focuses mainly on a structure that enables inference and logical conclusions.

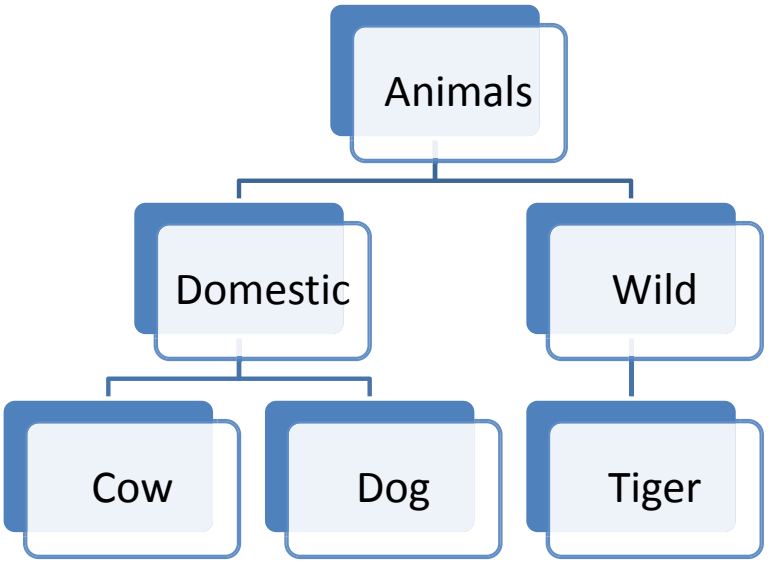


Figure 4.2: Graphical representation of taxonomy.

4.3 Difference between Ontology and taxonomy

Ontology	Taxonomy
Ontologies can be defined as an explicit specification of conceptualization.	Taxonomy is defined as a classification of elements in an ordered system that indicates natural relationships.
Ontologies generally include descriptive terms, in order to specify the vocabulary of the node contents.	Taxonomies generally do not include descriptive keywords for each item.
Ontologies represent a very complex form of interconnection.	Taxonomies are comparatively simpler and thrive to organize a subject in a particular way.
Relationships in ontology adhere to subjective interpretation, thus, triggering inference.	Taxonomies tend to be quite lenient about the kind of relationship that exists between parents and child node in the classification tree.

4.4 Types of Ontology

Several types of classifications are proposed for ontologies based on the characteristic of ontology's components. Explained below are few of the important types of classification, proposed for ontology.

4.4.1 Classification according to purpose

Application ontology

Application ontology is used in specific applications, in which a reasoner is implemented based on ontology.

Reference ontology

Reference ontology is used during development of applications, for mutual understanding and explanation between agents belonging to different communities, for establishing consensus in a community that has to adopt a new term or simplify a term for explaining the meaning of it to somebody new to the community.

4.4.2 Classification according to expressiveness

Heavyweight ontology

Heavyweight ontologies are heavily axiomatized and thus, represent an ontological commitment explicitly. The purpose of the axiomatization is to exclude terminological and conceptual ambiguities due to unintended interpretations.

Lightweight ontology

Lightweight ontologies are simple taxonomic structures of primitive or composite terms together with associated definitions. They are hardly axiomatized as the intended meaning of the terms used by the community is more or less known in advance by all members and the ontology can be limited to those structural relationships among the terms that are considered as relevant.

4.4.3 Classification according to specificity

Generic ontology

The concepts defined by this layer are considered to be generic across many fields. Typically, generic ontologies define concepts such as state, event, process etc.

Core ontology

Core ontologies define concepts which are generic across a set of domains. Therefore, they are situated in between the two extremes of generic and domain ontologies. The borderline between generic and core ontologies is not clearly defined because there is no exhaustive enumeration of fields and their conceptualizations. However, the distinction is intuitively meaningful and useful for building libraries.

Domain ontology

Domain ontologies express conceptualizations that are specific for a specific universe of discourse. The concepts in domain ontologies are often defined as a specialization of concepts in the generic and core ontologies. The borderline between core and domain ontologies is not clearly defined because core ontologies intend to be generic within a domain. Thus, it is usually hard to make a clear cut between generic and core as well as between core and domain ontologies. A concept such as a software component would be placed in core ontology for application servers.

4.5 Why to develop Ontology?

Some of the reasons to develop ontology are:

- To share a common understanding of the structure of information among people or software agents.
- To enable reuse of domain knowledge.
- To make domain assumptions explicit.
- To separate domain knowledge from the operational knowledge.
- To analyze domain knowledge.

Sharing common understanding of the structure of information among people or software agents:

This is one of the most common goals in developing ontologies (Musen 1992; Gruber 1993). For example, suppose that several different web sites contain medical information or provide medical e-commerce services. If these web sites share and publish the same underlying ontology of the terms they all use, then computer agents can extract and aggregate information from these different sites. The agents can use this aggregated information to answer user queries or as input data to other applications.

Enabling reuse of domain knowledge:

This was one of the driving forces behind the recent surge in ontology research. For example, models for many different domains must represent the notion of time. This representation includes the notions of time intervals, points in time, relative measures of time, and so on. If one group of researchers develops ontology in detail, others can simply reuse it for their domains. Additionally, if we need to build a large ontology, we can integrate several existing ontologies describing portions of the large domain. We can also reuse a general ontology, such as the UNSPSC ontology, and extend it to describe our domain of interest.

Making explicit domain assumptions

Explicit domain assumptions, underlying an implementation, make it possible to change the assumptions easily, if our knowledge about the domain changes. Hard-coding the assumptions about the world in programming-language code, makes these assumptions not only hard to find and understand, but also hard to change, in particular for someone without programming expertise. In addition, explicit specification of domain knowledge is useful for new users who must learn the meaning of terms in the domain.

Separating the domain knowledge from the operational knowledge

We can describe a task of configuring a product from its components, according to a required specification and implement a program that does this configuration independent of the products and components themselves (McGuinness and Wright 1998). We can then develop ontology of PC-components and characteristics and apply the algorithm to configure made-to-order PCs. We can also use the same algorithm to configure elevators if we feed elevator component ontology to it (Rothenfluh et al. 1996).

Analyzing domain knowledge

Analyzing domain knowledge is possible once a declarative specification of the terms is available. Formal analysis of terms is extremely valuable while attempting to reuse existing ontologies and while extending them (McGuinness et al. 2000). Often ontology of the domain is not a goal by itself. Developing ontology is akin to defining a set of data and their structure, for other programs to use. Problem-solving methods, domain-independent applications, and software agents use ontologies and knowledge bases built using ontologies as data.

4.6 Ontology development life-cycle

While building an ontology, the following questions will be helpful, in order to understand where to start and stop the ontology development process.

- What are the activities involved in the ontology development process?
- What is the goal of each activity?
- When should I carry out each activity?
- What is the relationship of one activity with the others?
- Where can I find ontologies with the goal of reusing them?
- How can I build the ontology for my application?
- Do I need a single ontology or an ontology network?

The different stages involved in the development life-cycle of ontology are listed below:

Identify the purpose and scope:

Every project, projected to develop using ontology as the base, will have a scope. This can be well-understood by comparing this stage with the requirement-analysis stage in application development. Developing a requirement specification for the ontology by identifying the intended scope and purpose of the ontology is the initial stage of ontology development life-cycle. A well-characterized requirement specification is important to the design, evaluation and re-use of ontology.

Knowledge Acquisition:

Knowledge Acquisition can be defined as the process of acquiring domain knowledge, using which ontology can be built. Different possible scenarios are identified and clubbed together. Informal competency questions are formed using it.

Conceptualization:

Conceptualization can be defined as a process of identifying the key concepts that exist in the domain, their properties and the relationships that hold between them; identifying natural language terms to refer to such concepts, relations and attributes, and structuring domain knowledge into explicit conceptual models. This is the process touched upon, in the beginning of this chapter, where the concepts and relationships describing the domain were captured. The ontology is usually described using some informal terminology. Gruber suggests writing down the lists of the concepts to be included in the ontology and exploring other ontologies to re-use all or part of their conceptualizations and terminologies.

Encoding:

Encoding is the process of representing the conceptualization in some formal language, e.g. frames, object models or logic. This includes the creation of formal competency questions in terms of the terminological specification language chosen (usually first order logic).

Integrating:

Integrating means to use or specialize in an existing ontology. This task is frequently hindered by the inadequate documentation of existing ontologies, notably their implicit assumptions. Using a generic ontology, gives a deeper definition of the concepts in the chosen domain.

Documentation:

Documentation includes informal and formal complete definitions, assumptions and examples that are essential to promote the appropriate use and re-use of ontology. Documentation is important for defining the exact meaning of terms within the ontology.

Evaluation:

Evaluation is determining the appropriateness of ontology for its intended application. Evaluation is done pragmatically, by assessing the competency of the ontology to satisfy the requirements of its application, including determining the consistency, completeness and conciseness of an ontology. Conciseness implies an absence of redundancy in the definitions of ontology and an appropriate granularity.

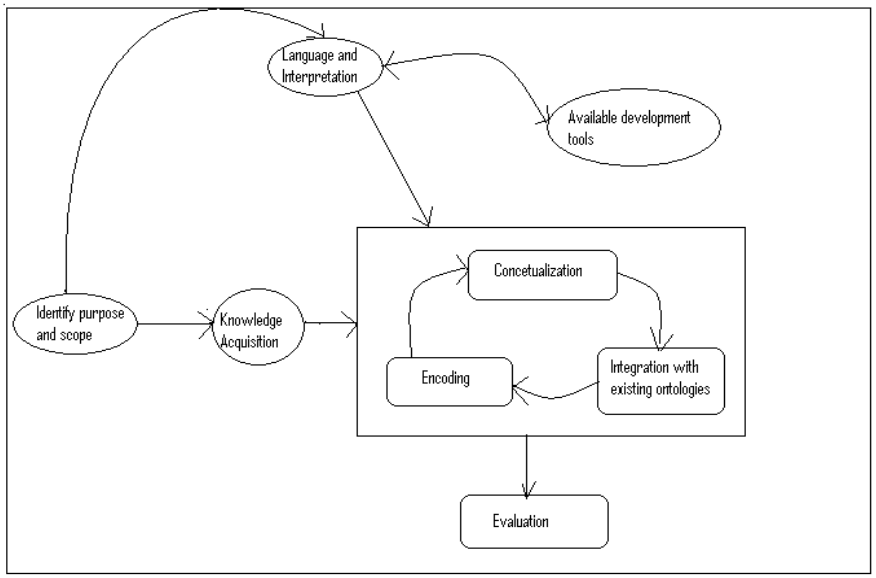


Figure 4.3: Ontology development life-cycle

4.7 Ontology Usage

The following application-independent lifecycle activities are performed at runtime to support ontology:

Ontology Population:

To populate the knowledge base (KB), instances may be collected from the user, e.g. via forms. A substantial overhead may be imposed on the user when all instances of data have to be created manually. This burden can be alleviated by a (semi)-automatic population of the KB. A part of this knowledge-creation step, also involves the manipulation and deletion of instances.

Cleansing and Fusion:

Automatically extracted knowledge cannot be assumed to have the desired quality. Enhancing the instance of data may include identification and merging of conceptually identical instances that are differently labeled (object identification).

Ontological Commitments

In order to share the knowledge amongst the agents, an agreement must exist on the topics which are being communicated. This raises the issue of ontological commitment. Ontological commitments allow a number of agents to meaningfully communicate about a domain without necessarily operating on a globally shared theory. In the context of multiple agents, a common ontology serves as a knowledge-level specification of the ontological commitments of a set of participating agents. A common ontology defines the vocabulary with which queries and assertions are exchanged among the agents, thereby providing the means to bridge the semantic gap that exists between the lexical representations of information and its non-lexical conceptualization.

4.7.1 Ontology Learning

Ontology learning is defined as the set of methods and techniques used for building ontology from scratch, enriching, or adapting an existing ontology in a semi-automatic fashion using distributed and heterogeneous knowledge and information sources, allowing reduction in time and effort needed in the ontology development process. Though a fully automatic acquisition of knowledge remains inaccessible, the overall process is considered as semi-automatic, i.e. human intervention is necessary in some parts of the learning process. New concepts are identified using natural language analysis techniques over the resources previously identified by the user. The resulting ontology is pruned and then focused on a specific domain by means of several approaches based on statistics. Finally, the relation between the concepts is established by applying learning methods.

4.7.2 Ontology Alignment

Ontology alignment consists of establishing different kinds of mappings (or links) between two ontologies, hence preserving the original ontologies. Ontology merging proposes to generate a unique ontology from the original ontologies. We will assume that a mapping between ontologies means rewriting a set of rules that associates terms and expressions defined in source ontology with terms and expressions of a target ontology.

4.8 Advantages of Ontology

A good ontology offers a composite suite of benefits that are not available in taxonomies, relational database schema, or other standard ways, to structure information. Some of these benefits are:

- **Ontologies promote coherent navigation** by enabling the movement from concept to concept in the ontology structure.
- **They have flexible entry points,** because any specific perspective in the ontology can be traced and related to all of its associated concepts; there is no specific set structure or manner for interacting with the ontology.
- **Connections** highlight related information and aid prompt discovery without requiring prior knowledge of the domain or its terminology.
- **Ability to represent any form of information**, including unstructured (say, documents or text), semi-structured (say, XML or Web pages) and structured (say, conventional databases) data.
- **Inferencing,** whereby by specifying one concept (say, mammals) one knows that we are also referring to a related concept (say, that mammals are a kind of animal).
- **Concept matching**, which means that even though we may describe things somewhat differently, we can still match to the same idea (such as glad or happy both referring to the concept of a pleasant state of mind).
- Thus, this means that we can also **integrate external content** by proper matching and mapping of these concepts.
- **Reasoning**, which is the ability to use the coherence and structure it to inform questions of relatedness or to answer questions; this latter benefit is more related to machine learning or artificial intelligence, and is not generally expressed in simpler, standard ontologies.

4.9 Limitations of Ontology

Though ontology is contributing to the progress of Semantic Web, it also has some limitations.

- Ontology makes the abstract model of a particular domain based on set of data and structures but does not define the boundaries of the model.
- Size of ontology varies with respect to the number of classes and instances; if the number of instances is increased to a large extent then it becomes very hard to manage manually, and currently there is no mechanism to manage it automatically.
- Manual ontology generation process sometimes becomes very complex and time consuming, especially while dealing with a large amount of data.

5 RDF and SPARQL

Objective:

Once data is stored in a database, it's an obvious fact that it must be queried, in order to retrieve the data and use it in an application. This chapter discusses about the data model of graphs, i.e. RDF, and SPARQL which is a querying language for RDF.

5.1 Introduction to RDF

RDF stands for Resource Description Framework. RDF is a family of World Wide Web Consortium (W3C) specifications, originally designed as a metadata data model. It is used as a general method for conceptual description or modeling of information that is implemented in web resources, using a variety of syntax notations and data serialization formats.

5.1.1 Explanation

RDF is the format in which the graph data model, used by the Semantic Web to store data, is written. RDF is a general method to decompose knowledge into small pieces, with some rules about the semantics, or meaning of those pieces. The point is to have a method so simple that it can express any fact, and yet so structured that computer applications can do useful things with knowledge expressed in RDF. It is termed as 'method' in particular rather than format, because one can write down those pieces in any number of ways and still preserve the original information and structure, just like how one can express the same meaning in different human languages or implement the same data structure in multiple ways.

RDF can be defined using three simple rules:

- A fact is expressed as a triple of the form (Subject, Predicate, and Object). It's almost like an English sentence.
- Subjects, predicates, and objects are names of entities, whether concrete or abstract, in the real world. Names are either:
  1) Global and refer to the same entity in any RDF document in which they appear, or
  2) Local and the entity it refers to cannot be directly referred, outside the RDF document.
- Objects can also be text values, called literal values.

5.1.2 Revisiting XML

XML stands for extensible Markup Language. XML is a specification for computer-readable documents. Markup means that certain sequences of characters in the document contain information indicating the role of the document's content. The markup describes the document's data layout and logical structure and makes the information self-describing, in a sense. It takes the form of words between the tags – for example, <name> or <age>.
In this aspect, XML looks very much like the well-known language HTML.

Example:
<?xml version="1.0"?>
<employees>
List of persons in company:
<person name="Nandini">
<phone>123456789</phone>
</person>
</employees>

XML does not imply a specific interpretation of the data. Of course, on account of the tag's names, the meaning of the previous piece of XML seems obvious to human users, but it is not formally specified.

The only legitimate interpretation is that XML code contains named entities with sub-entities and values; that is, every XML document forms an ordered, labeled tree. This generality is both XML's strength and its weakness. You can encode all kinds of data structures in an unambiguous syntax, but XML does not specify the data's use and semantics. The parties that use XML for their data exchange must agree beforehand on the vocabulary, its use and its meaning.

5.1.2.1 DTD and XML Schema

DTD stands for Document Type Definitions. DTD and XML Schemas partly answered the drawback of XML. Although DTDs and XML Schemas do not specify the data's meaning, they do specify the names of elements and attributes (the vocabulary) and their use in documents. Both are mechanisms with which you can specify the structure of XML documents. You can then validate specific documents against the structure prescription specified by a DTD or an XML Schema. DTDs specify the allowed nesting of elements, the element's possible attributes, and the locations where normal text is allowed. For example, a DTD might prescribe that every 'person' element must have a "name" attribute and may have a child element called 'phone' whose content must be text. A DTD's syntax looks a bit awkward, but it is actually quite simple.

XML Schemas are a proposed successor to DTDs. The XML Schema definition is still a candidate recommendation from the W3C (World Wide Web Consortium), which means that, although it is considered stable, it might still undergo small revisions. XML Schemas have several advantages over DTDs. First, the XML Schema mechanism provides a richer grammar for prescribing the structure of elements. For example, you can specify the exact number of allowed occurrences of child elements, you can specify default values, and you can put the elements in a choice group, which means that exactly one of the elements in that group is allowed at a specific location. Second, it provides data typing. A final difference from DTDs is that XML Schema prescriptions use XML as their encoding syntax. This simplifies tool development, because both the structure prescription and the prescribed documents use the same syntax. The XML Schema specification's developers exploited this feature by using an XML Schema document to define the class of XML Schema documents. After all, because an XML Schema prescription is an XML application, it must obey rules for its structure, which can be defined by another XML Schema prescription. However, this recursive definition can be a bit confusing.

XML Entities

An XML entity can play several roles, such as a placeholder for repeatable characters (a type of shorthand), a section of external data (e.g., XML or other), or as a part of a declaration of elements.
For example, suppose that a document has several copyright notices that refer to the current year. Then it makes sense to declare an entity <!ENTITY thisyear "2007">.

Then, at each place where the current year needs to be included, we can use the entity reference &thisyear; instead. That way, updating the year value to "2008" for the whole document will only mean changing the entity declaration.

5.1.3 Why bother about XML in RDF

XML provides syntax to encode data**.** It means that the resource description framework is a mechanism to tell something about the data. As the name indicates, it is not a language, but a model for representing data about 'things on the web.' This type of data about data is called metadata. The 'things' are resources in RDF vocabulary. RDF's basic data model is simple: besides resources, it contains properties and statements. A property is a specific aspect, characteristic, attribute, or relation that describes a resource. A statement consists of a specific resource with a named property plus that property's value for that resource. This value can be another resource or a literal value, such as free text. Altogether, an RDF description is a list of triples: an object (a resource), an attribute (a property), and a value (a resource or free text).

Note:

People often feel that RDF and XML are the same. But the fact is that RDF is a data model which can be represented in XML (RDF/XML). RDF is a graph data model that makes use of URIs while XML is a tree data model and doesn't care about URIs.

5.1.4 RDF Schema

RDF Schema (RDFs) is an extension of RDF vocabulary to allow taxonomical description of classes and their properties. It also extends the definitions of some of the elements of RDF, for example it sets the domain and range of properties and relates the RDF classes and properties into taxonomies using the RDFS vocabulary.

5.2 RDF graph

RDF can be defined as a 2-Dimensional concept. It can be either perceived as a graph or as a set of statements. Let us begin with the former one.

Going by the classical definition, a graph is a representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by mathematical abstractions called vertices, and the links that connect some pairs of vertices are called edges.

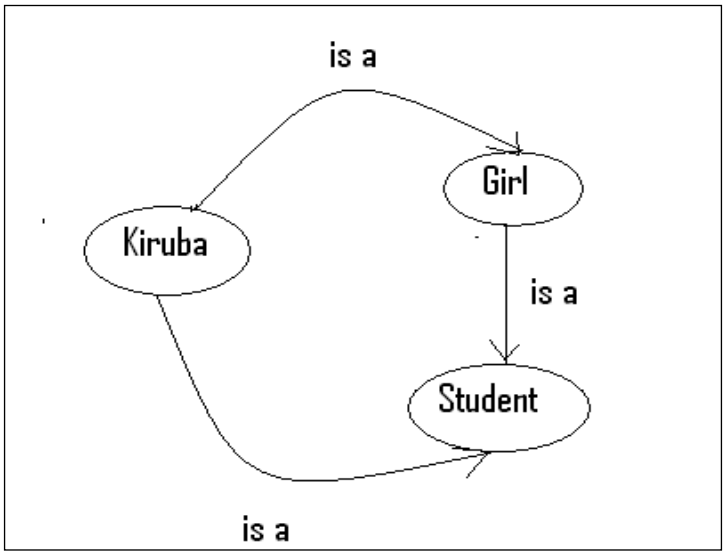


Figure 5.1: A simple graph

Refer figure 5.1. The name at the start of the arrow is the statement's subject, the name at the end of the arrow is the statement's object, and the name that labels the arrow is the predicate. RDF as a graph expresses exactly the same information as RDF written as triples, but the graph form makes it easier for us to see the structure in the data. Hence we can conclude:

Subject = (Kiruba, Girl)
Object = (Girl, Student)
Predicate = (is a)

5.2.1 RDF Basics

Resources

We can think of a resource as an object, a 'thing' we want to talk about. Resources may be authors, books, publishers, places, people, hotels, rooms, search queries, and so on. Every resource has a URI, a Uniform Resource Identifier. A URI can be a URL (Uniform Resource Locator, or Web address) or some other kind of unique identifier (note that an identifier does not necessarily enable access to a resource). URI schemes have been defined not only for web locations, but also for such diverse objects as telephone numbers, ISBN numbers, and geographic locations. There has been a long discussion about the nature of URIs, even touching philosophical questions (for example, what is an appropriate unique identifier for a person?), but we will not go into detail here. In general, we assume that a URI is the identifier of a Web resource.

Properties

Properties are a special kind of resources. They describe relations between resources, for example 'written by', 'age', 'title', and so on. Properties in RDF are also identified by URIs (and in practice by URLs). This idea of using URIs to identify 'things' and the relations between them is quite important.
This choice gives us in one stroke a global, worldwide, unique naming scheme. The use of such a scheme greatly reduces the homonym problem that has plagued distributed data representation until now.

Statements:

Statements assert the properties of resources. A statement is an object, attribute-value triple, consisting of a resource, a property, and a value. Values can either be resources or literals. Literals are atomic values (strings), the structure of which we do not discuss further.

5.3 Constructing RDF

RDF provides a general, flexible method to decompose any knowledge into small pieces, called triples, with some rules about the semantics (meaning) of those pieces. The foundation is laid by breaking the knowledge into basically what's called a labeled, directed graph, as discussed in the previous section.

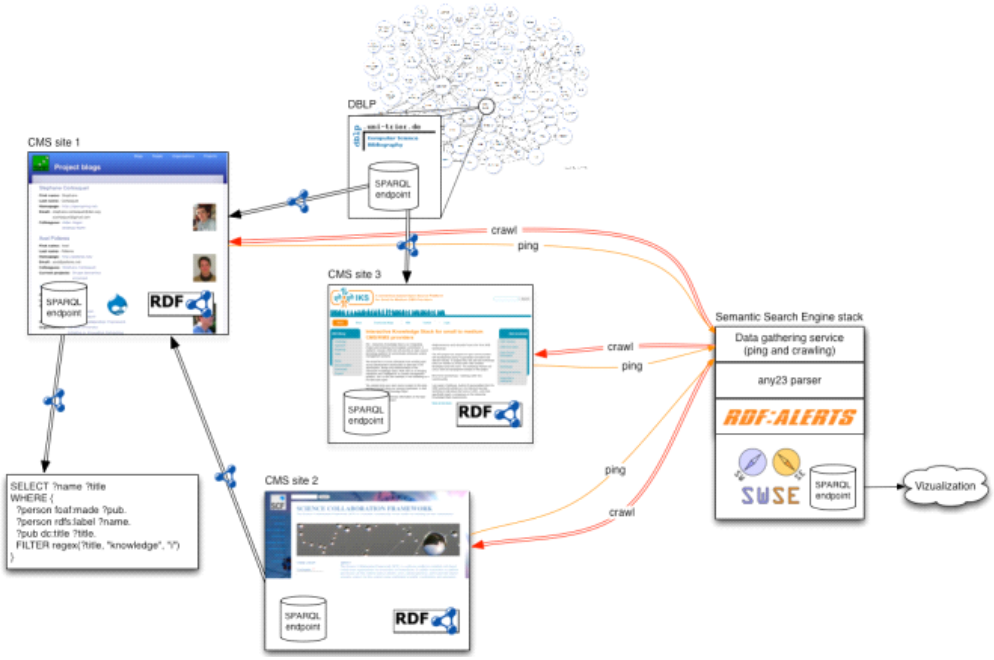


Figure 5.2: System implemented using RDF.

5.3.1 RDF Triples

Let us start with the standard syntax of RDF triples. Stated below is the standard syntax-

<rdf:Description rdf:about="subject">
<predicate rdf:resource="object" />
<predicate>literal value</predicate>
<rdf:Description>

In order to understand this concept properly, let us start with an example. Consider the following example,

Example 1:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:feature="http://www.bookboon.com/cartoon-features#">
<rdf:Description rdf:about="http://www.bookboon.com/cartoon#mickey-mouse">
<feature:character rdf:resource="http://www.bookboon.com/character#funny"/>
</rdf:Description>
</rdf:RDF>
```

Explanation:

RDF triple

In example 1, the RDF between the <rdf: Description> tags is called an RDF statement, or an RDF triple. In RDF triple, as the name suggests, the statement is broken into three parts:

- Subject
- Object
- Predicate of the statement

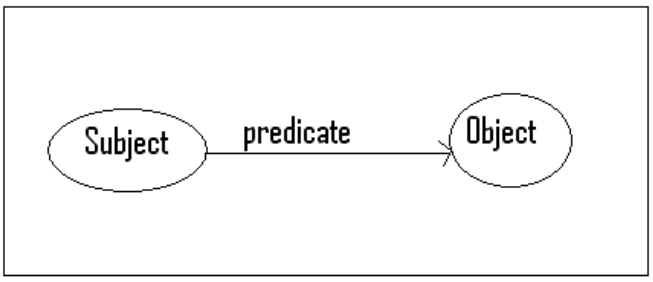


Figure 5.3: Graphical representation of triples

Refer example 1,

The namespace "**//www.w3.org/1999/02/22-rdf-syntax-ns#**" that we find, is the standard W3.org namespace. This namespace tells any machine reader that the enclosing document is an RDF document, and that the rdf:RDF tag resides in this namespace. This namespace, and the RDF node, form the root of all RDF documents.

Constructing a statement in RDF:

An RDF document can contain more than one statement. The rdf:Description tag describes the **subject** and gives it a unique identifier namely, **http://www.bookboon.com/cartoon#mickey-mouse**. Hence, here the subject is 'Mickey-mouse'.

Constructing a predicate in RDF:

RDF statements describe the characteristics of their subjects using properties, or predicates in RDF terminology. The subject has a property with name **feature:character** which has the literal value 'funny', i.e. the cartoon character Mickey-mouse has a funny characteristic. So, here the predicate is 'character'.

Constructing an object in RDF:

The **feature** i.e. character, in

"**<feature:character rdf:resource="http://www.bookboon.com/character#funny**"/>", is referring to the subject (ID) of another statement. It implies that objects in RDF can refer the subjects of other statements. The object here is 'funny'.

In graphical format Example 1 can be represented as follows:

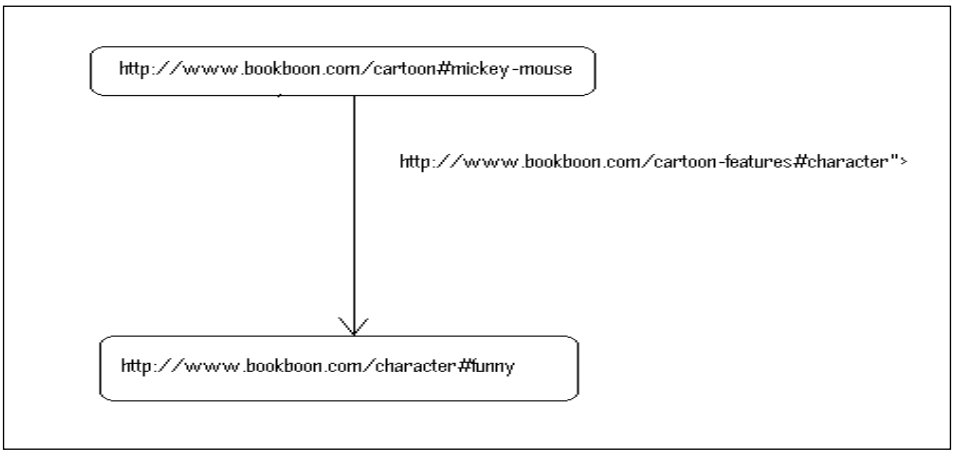


Figure 5.4: RDF-Graph for example 1

5.4 Introduction to SPARQL

SPARQL stands for 'SPARQL Protocol and RDF Query Language'. Like the tables of a relational database are queried using SQL, the triples of RDF data are queried using SPARQL. SPARQL is a query language designed specifically to query RDF databases. SPARQL queries are sent from a client to a service known as a SPARQL endpoint using the HTTP protocol. The interaction between the client and the endpoint is defined in a machine-friendly protocol that is not intended to be interpreted by humans, so use of SPARQL requires an interface that allows the user to enter the queries and to display the results in a meaningful way. As with traditional database languages such as SQL, interfaces are commonly constructed so that the queries are constructed and launched through forms that do not require the human user to have any knowledge of RDF and SPARQL.

In addition to the language, W3C has also defined:

- The **SPARQL Protocol** for RDF specification: it defines the remote protocol for issuing SPARQL queries and receiving the results.
- The **SPARQL Query Results XML** Format specification: it defines an XML document format for representing the results of SPARQL queries.

5.5 SQL

SQL stands for Structured Query Language. SQL is a computer language for storing, manipulating and retrieving data stored in relational database. A Computer engineering student will find this term too familiar. This is because, most of the existing applications use SQL to query database. SQL is the standard language for a Relational Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server use SQL as the standard database language. SQL is followed by a unique set of rules and guidelines called Syntax. All the SQL statements start with a keyword like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW and the entire statements end with a semicolon.

Let us discuss some of the important keywords in SQL that will be useful in understanding SPARQL:

CREATE

Creating a basic table involves naming the table and defining its columns and each column's data type. The SQL **CREATE TABLE** statement is used to create a new table.

Example:

```
Create table cust (

Emp_no int not null,

Name varchar(25)

);
```

SELECT

SQL **SELECT** statement is used to fetch the data from a database table which returns data in the form of result table. These result tables are called result-sets.

Example:

```
Select * from customer_information;
```

This will retrieve all the data from the table "customer_information".

INSERT

SQL **INSERT** statement is used to insert the data into a database table.

Example:

Insert into customer_information values (A, B, C);

This will insert the data into the table "customer_information".

SPARQL Vs SQL

Many people ask what can be done using SPARQL that can't be done using SQL. Both, SQL and SPARQL, give access to the user to create, combine, and consume structured data. SQL does this by accessing tables in relational databases, and SPARQL does this by accessing a web of Linked Data. RDF captures both entity attributes and relationships between entities, as statements of the form '**entity1** has **property A** relationship to **entity2'** using a language called Turtle. We can say that there is a person named "Mickey" with an address in Disney, USA, as follows:

<PersonA>a<Person>.
<AddressB>a<Address>.
<PersonA><Person#fname>"Mickey".
<AddressB><Address#city>"Disney".
<PersonA><Person#addr><AddressB>.
<AddressB> <Address#state> "USA" .

Also, we can have another person, "Pluto", but we won't say anything about her address because we don't know it. Hence, we represent as follows:

<PersonF>a<Person>.
<Person> <Person#fname> "Pluto".

The terms used in the statements above are relative URLs in angle brackets (<>s), literals in quotation marks (""s) and the keyword 'a' which is just a shortcut for the URL used to identify a 'has type' relationship. There is no concept in RDF corresponding to SQL's NULL as there is no RDF requirement corresponding to SQL's structural constraint that every row in a relational database must conform to the same schema. The object of one assertion, e.g.<AddressB> above, may be the subject or object of other assertions. In this way, a set of RDF statements connects up to create a "graph" (in the mathematical sense). You will frequently hear the term 'RDF graph'. These graphs may be cyclic, for example, stating that Mickey lives in someplace, for which, he is also the owner:

<PersonC><Person#homeAddress><AddressK>.
<PerconC><Person#fname>"Mickey".
<AddressK><Address#owner><PersonC>.

The examples above illustrate some of the structural similarities and differences between RDF and relational data. A core philosophical difference is that RDF is a post-Web language; that is, it allows one to use web identifiers for the entities we want to describe, and for the attributes and relationships we use to describe them. If I trust the publishers not to lie to me, I can merge information from different parties trivially.

Finally, consider the following example:

- SELECT Person.fname, Address.city FROM Person, Address WHERE Person.addr=Address.ID AND Address.state="Mumbai"

Conceptually, we are selecting a list of attributes from a set of tables, where certain constraints are met. These constraints capture the relationships implicit in the scheme, Person.addr=Addresses.ID, and the selection criteria, e.g. Address.state="Mumbai".

A SPARQL query of the same data could look like –

```
SELECT?name?city
WHERE{

?who<Person#fname>?name;
<Person#addr>?adr.
?adr<Address#city>?city;
 <Address#state>“Mumbai”
}
```

For better or worse, SPARQL reuses some key words familiar to SQL users: SELECT, FROM, WHERE, UNION, GROUP BY, HAVING and most aggregate function names.

5.6 Constructing a SPARQL query

The SPARQL query language is based on matching graph patterns. The simplest graph pattern is the triple pattern, which is like an RDF triple but with the possibility of a variable instead of an RDF term in the subject, predicate, or object positions. Combining triple patterns give a basic graph pattern, where an exact match to a graph is needed to fulfill a pattern.

As a simple example, consider the following query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?c
WHERE
{
?c rdf:type rdfs:Class .
}
```

This query retrieves all triple patterns where the property is rdf:type and the object is rdfs:Class. In other words, this query when executed will retrieve all classes.

Format for constructing a SPARQL query:

Figure 5.1: Format of SQL query

Example:

PREFIX school-info: <http://education.data.gov.india/def/school/>

SELECT ?name WHERE {

?school a school-info:School.

?school school-info:establishmentName ?name.

?school school-info:districtAdministrative <http://statistics.data.gov.uk/id/local-authority-district/Mumbai>.

}

ORDER BY ?name

This query, when executed, will return the names of all the schools in India, in administrative district "Mumbai", and orders the results in alphabetical order.

The familiarity with SQL will be very useful in understanding the format and construction of SPARQL. The **SELECT** statement requests the variable **?name** to be returned. **?name** returns all the names of the schools which match the three search patterns given in the query.

6 Protégé

Objective:

Protégé is the most famous tool in the Semantic Web community to build ontologies. This chapter will discuss in detail, the method to create an ontology using Protégé, and the ways to explore the concept of classes and individuals in the ontology.

6.1 Introduction to Protégé

Protege is an open-source tool developed at Stanford Medical Informatics. It has a community of thousands of users. Although the development of Protege has historically been mainly driven by biomedical applications, the system is domain-independent and has been successfully used for many other application areas as well. Like most other modeling tools, the architecture of Protege is cleanly separated into a 'model' part and a 'view' part. Protege's model is the internal representation mechanism for ontologies and knowledge bases. Protege's view components provide a user interface to display and manipulate the underlying model. The protege's model is based on a simple, yet flexible metamodel, which is comparable to object-oriented and frame-based systems. Basically, it can represent ontologies consisting of classes, properties (slots), property characteristics (facets and constraints), and instances.

Protege provides an open Java API to query and manipulate models. An important strength of Protege is that the Protege metamodel itself is a Protege ontology, with classes that represent classes, properties, and so on. For example, the default class in the Protege base system is called STANDARD-CLASS, and has properties such as :NAME and :DIRECT-SUPERCLASSES. This structure of the metamodel enables easy extension and adaption to other representations. For example, this metamodel is extended to handle UML and OWL. Using the views of Protege's user interface, ontology designers basically create classes, assign properties to the classes, and then restrict the properties' facets at certain classes. Using the resulting ontologies, Protege is able to automatically generate user interfaces that support the creation of individuals (instances). For each class in the ontology, the system creates one form with editing components (widgets) for each property of the class.

For example, for properties that can take single string values, the system would, by default, provide a text field widget. The generated forms can be further customized with Protege's form editor, where users can select alternative user interface widgets for their project. In addition to the predefined library of user interface widgets, Protege has a flexible architecture that enables programmers to develop custom-tailored widgets, which can then be plugged into the core system. Another type of plugin supports full-size user interface panels (*tabs*) that can contain arbitrary other components.

In addition to the collection of standard tabs for editing classes, properties, forms, and instances, a library of other tabs exists that perform queries, access data repositories, visualize ontologies graphically, and manage ontology versions. Protégé currently can be used to load, edit and save ontologies in various formats, including CLIPS, RDF, XML, UML and relational databases. Recently, support for OWL was added. Since ontologies played an important role in Semantic Web applications, it was straightforward to take an existing ontology development environment as a starting point. Extensions to Protege can benefit from the generic services provided by the core platform, such as an event mechanism, undo capabilities, and a plugin mechanism. By basing the OWL Plugin on top of Protege, we could also reuse Protege's client-server-based multi-user mode that allows multiple people to edit the same ontology at the same time. Protege also provides a highly scalable database back-end, allowing users to create ontologies with hundreds of thousands of classes.

Also, there is already a considerable library of plugins which can be either directly used in OWL or adapted to OWL with little effort. Furthermore, the fact that Protege is open-source also encourages plugin development. Last but not least, Protege is backed by a large community of active users and developers, and the feedback from this community proved to be invaluable for the development of the OWL Plugin. Our decision to base the OWL Plugin on Protege also had some risks. In order to be able to reuse as much of the existing Protege features as possible, we had to create a careful mapping between the Protege metamodel and OWL that maintains the traditional Protege semantics where possible. Furthermore, none of the generic Protege widgets and tabs is optimized for OWL, and not all of the editing metaphors for frame based systems are appropriate for OWL. In particular, OWL's rich description logics features such as logical class definitions required special attention. The following sections will show how we have addressed these issues.

6.1.1 How to develop Ontology?

Developing ontology in theoretic terms is already discussed in the previous chapter. The following steps will help to develop ontology practically:-

1. Define classes in the ontology
2. Arrange the classes in a subclass-superclass hierarchy
3. Define slots and describe allowed values for these slots
4. Fill-in the values of slots for instances

6.2 Files in Protégé:

When you use Protégé to create and edit ontologies you will generate at least two files:

- **Project file**
  A project file has an extension of .pprj. The project file stores information related to any interface customizations or editor options you have selected. In many cases, you don't have to send this file to colleagues with your ontology unless you have been customizing forms with the Forms Tab. Project files created with older versions of Protégé-OWL (especially prior to version 3.0) may not be compatible with your current version, in such case you may have to build a new project from the scratch.

- **Source file**
  A source file has an extension of .owl, .rdfs, or .rdf. This file is where your ontology classes, individuals and properties are defined. There may be several source files depending on how the ontology has been defined. If it is modular and has been created properly, Protégé-OWL will find and load all of the appropriate source files.

- **Classes**
  Ontology classes are very similar to classes in an object oriented program. Just like object oriented programming, classes in ontologies also form an hierarchy. You can view this hierarchy in the right-hand side panel, labeled as Asserted class hierarchy. The root class which is at the top of the inheritance hierarchy is Thing (this is true of all OWL ontologies).

6.2.1 How are Ontology Classes different from Object-oriented Classes?

While they are similar in many ways, one important difference to remember is that, an individual of ontology can belong to zero or more classes, in addition to any inherited class. There are ways to limit the classes an individual of one class can belong to but if such explicit information is absent then individuals can be members of, as few or as many classes as the ontology creator wants them to be.

- **Equivalent classes**
  This section describes other classes or groups that are equivalent to the selected class.

- **Superclasses**
  Superclasses are the parent class/classes of the selected class (since you can have multiple classes you can also have multiple superclasses).

- **Members**
  Members represent individuals which are members of a particular class. They can be added explicitly, or inferred later through reasoning.

- **Disjoint classes**
  They allow you to explicitly select the classes that members of the selected class cannot belong to. Most of the time, you do not have to explicitly mark classes as disjoint but it can be helpful if you are using some external reasoning or application to make the class definitions as explicit and exact.

  Browse through the classes provided and try to get a feel of the concepts they are trying to describe. After you are done looking at the classes go to the top menu and select Reasoner -> Fact++. This will turn on the reasoner to infer additional facts about the classes.

- **Object and Data properties**
  While attaching properties to classes, it makes sense to immediately provide statements about the domain and range of these properties. There is a methodological tension here between generality and specificity. It is useful to define the domain and range as narrowly as possible, so that it will be easy to detect potential inconsistencies and misconceptions in the ontology by spotting domain and range violations.

- **OWL properties represent relationships between two objects.**

  There are two main properties:

  1. Object properties link object to object.
  2. Data-type properties link object to XML Schema datatype or rdf:literal

Some other important properties are:

Annotation properties

This property can be used to add annotation information to classes, individuals, and properties.

Inverse properties:

1. Every object property may has a corresponding inverse property.
2. If some property links individual A to individual B, then its inverse property will link individual B to individual A.

Functional properties:

1. If a property is functional for a given individual, then only one individual can be related via this property.
2. Functional properties are also known as single valued properties.

Inverse functional properties:

If a property is inverse functional, then its inverse property is functional.

Transitive properties:

If a property P is transitive, and the property relates individual A to individual B, and also individual B to individual C, then we can infer that individual A is related to individual C via property P.

Symmetric properties:

If a property P is symmetric, and the property relates individual A to individual B, then individual B is also related to individual A via property P.

6.2.3 Some important terms

Cardinality:

Cardinality must be specified for as many properties as possible in order to indicate whether they are allowed or required to have a certain number of different values or not. Often, occurring cases are 'at least one value' (i.e., required properties) and 'at most one value' (i.e., single-valued properties).

Required values:

Often, classes are defined by virtue of a certain property having particular values, and such required values can be specified in OWL, using owl:hasValue. Sometimes, the requirements are less stringent, for instance-

A property may have some values from a given class (and not necessarily a specific value, owl:someValuesFrom).

Relational characteristics:

It is concerned with the relational characteristics of properties, i.e., symmetry, transitivity, inverse properties, functional values.

Property domain and ranges

Properties link individuals from the domain to individuals from the range. OWL uses domain and range as axioms in reasoning.

Property restrictions:

In OWL, properties are used to create restrictions. Restrictions are used to restrict the individuals that belong to a class.

There are five restrictions:

- Quantifier restrictions
- Existential quantifier
- Universal quantifier
- Cardinality restrictions
- hasValue restrictions

6.3 Instances

We use ontologies to organize sets of instances and there is a separate step to fill the ontologies with instances. Typically, the number of instances is larger than the number of classes in the ontology.

Ontologies vary in size from a few hundred classes to tens of thousands of classes; the number of instances varies from hundreds to hundreds of thousands, or even larger. Because of these large numbers, populating ontology with instances is typically not done manually. Often, instances are retrieved from legacy data sources such as databases. Another, often used technique is the automated extraction of instances from a text corpus.

6.3.1 Creating a sample project

Note:

There are two main ways of modeling ontologies:

- Frame-based
- OWL

Each has its own user interface.

1. Protege Frame editor: It enables users to build and populate ontologies that are frame-based, in accordance with OKBC (Open Knowledge Base Connectivity Protocol). It consists of –
   - Classes
   - Slots for properties and relationships
   - Instances of class

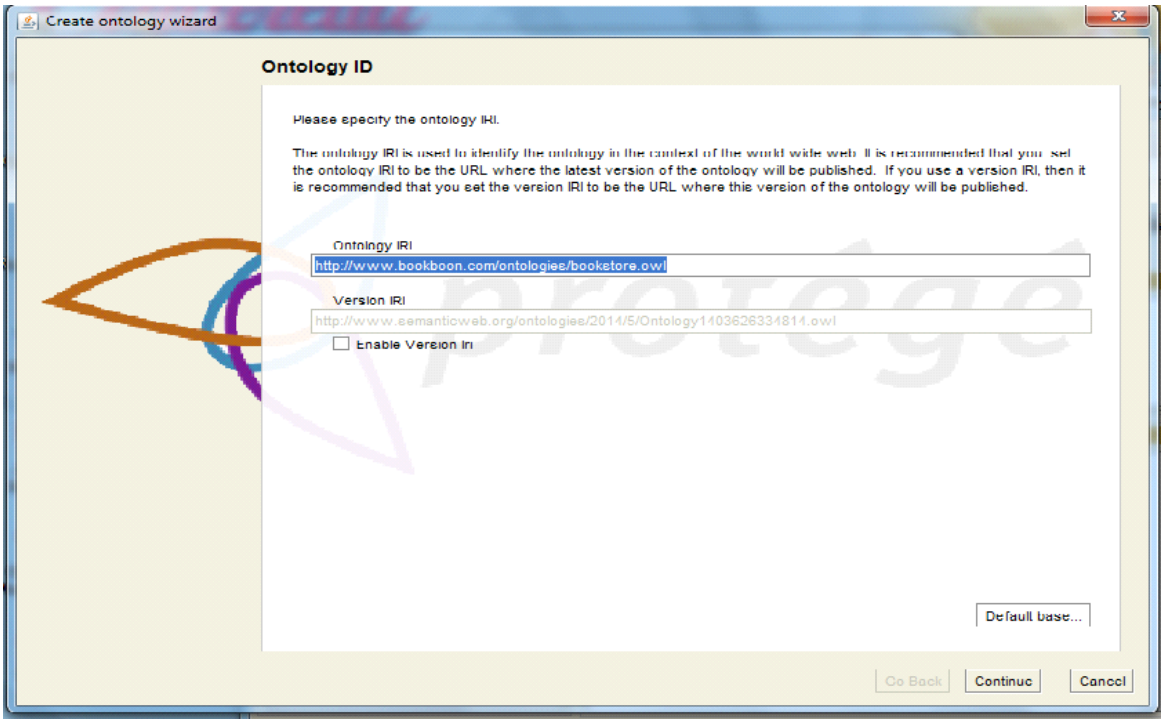
2. Protege OWL editor:
   It enables users to build ontologies for the Semantic Web. It consists of-
   - Classes
   - Properties
   - Instances
   - reasoning

Tool used: Protégé

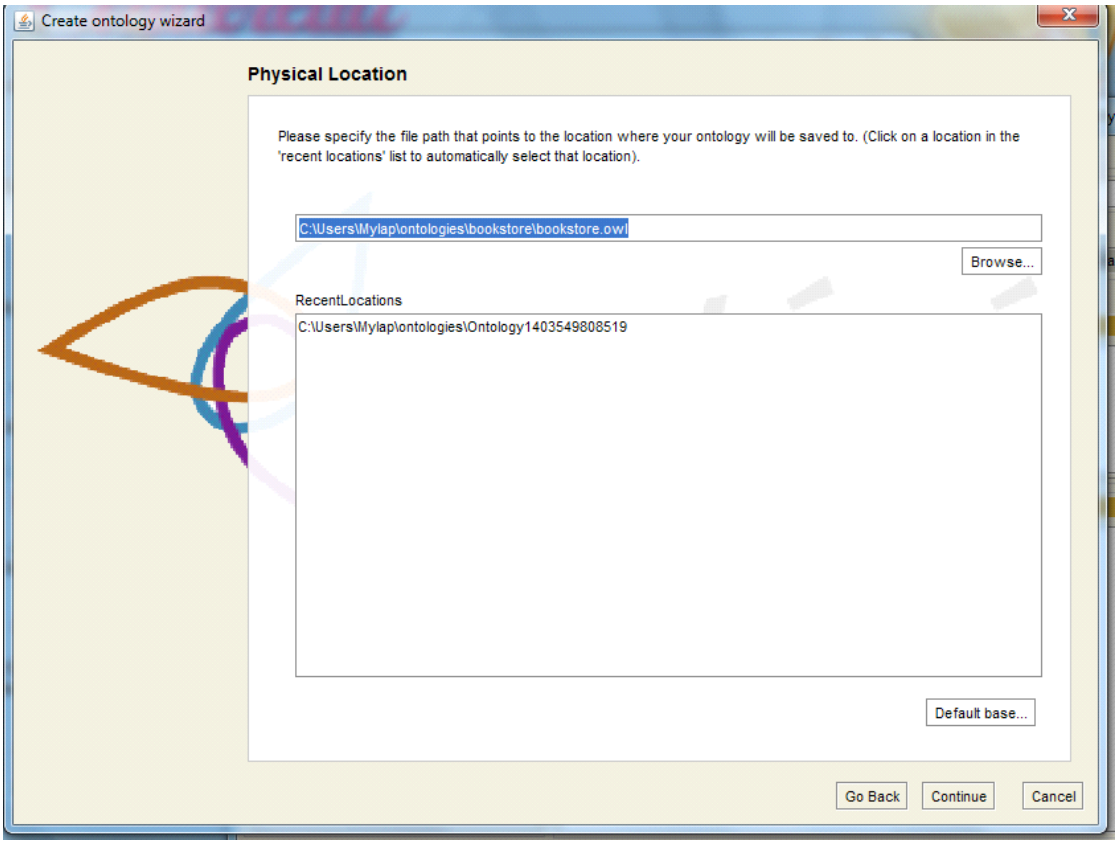
Step 1:

1. Start protégé
2. Go to File
3. Choose New
4. Enter the Ontology URI (**http://www.bookboon.com/ontologies/bookstore.owl**)
5. Then choose RDF/XML
6. Choose Properties View

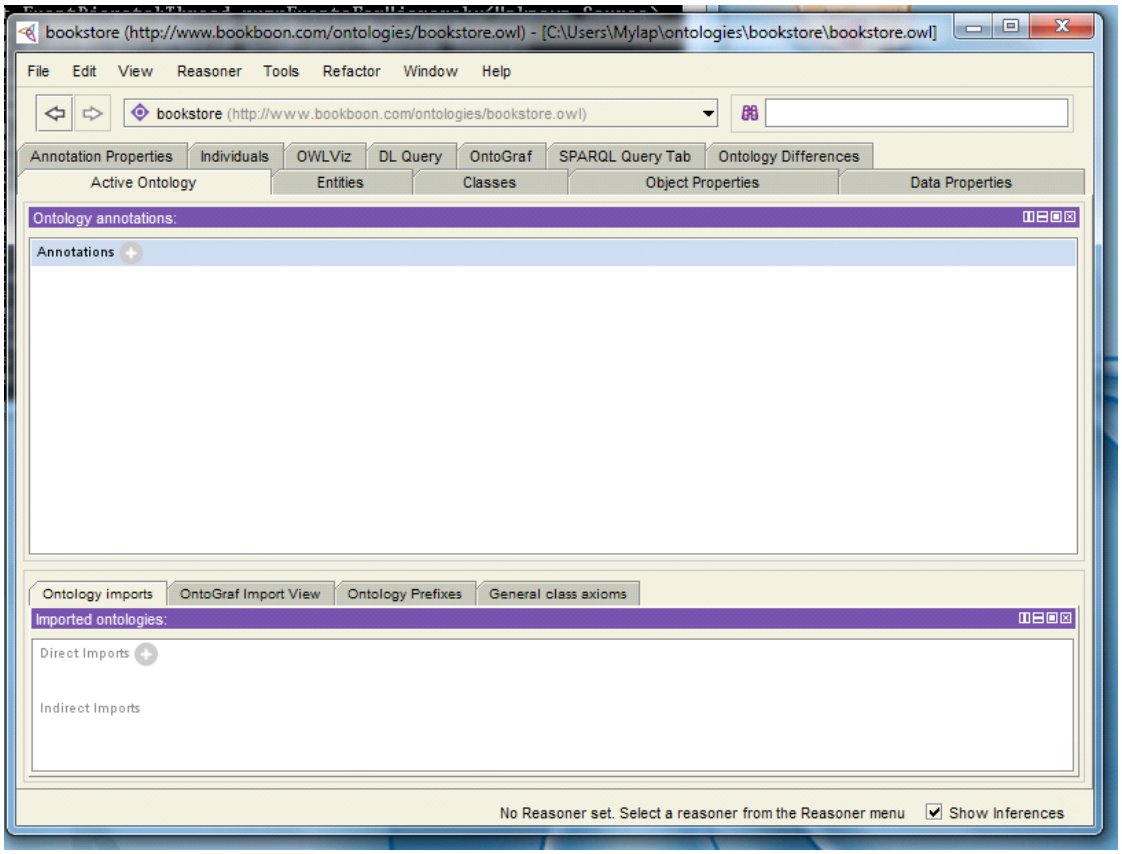
A new, empty Protégé-project has been created. Save it in your local file as bookstore.owl. Refer the following screenshots for better understanding:



Screenshot 6.1: Create Ontology Wizard



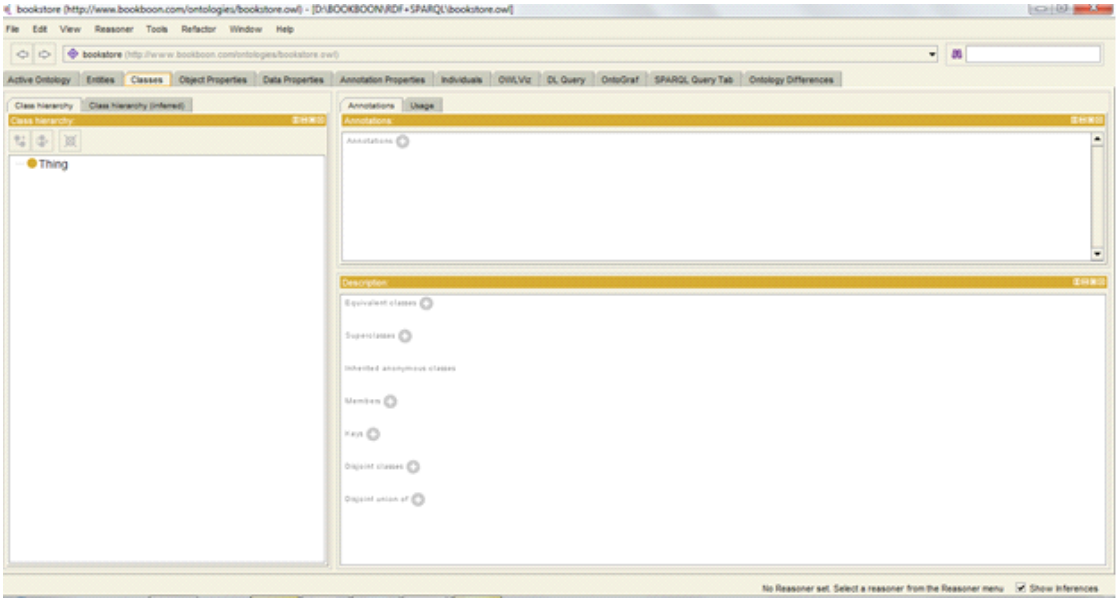
Screenshot 6.2: Create Ontology Wizard – Specifying the location to store the .owl file



Screenshot 6.3: Protégé interface

Step 2:

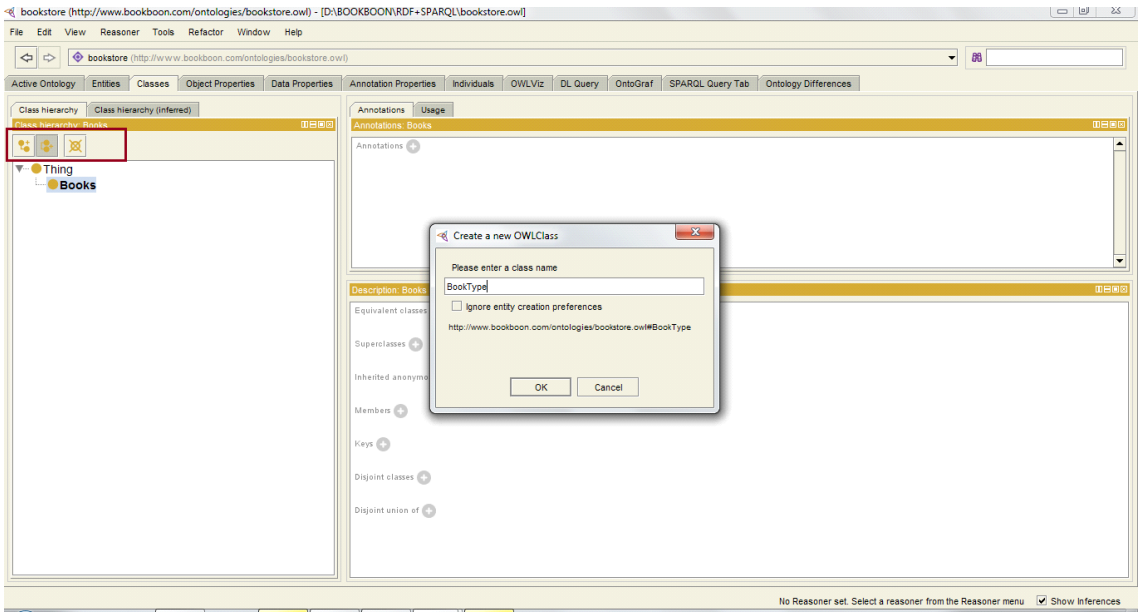
Go to Classes tab. You will find that the empty class tree contains one class called owl:Thing, which is the superclass of everything.



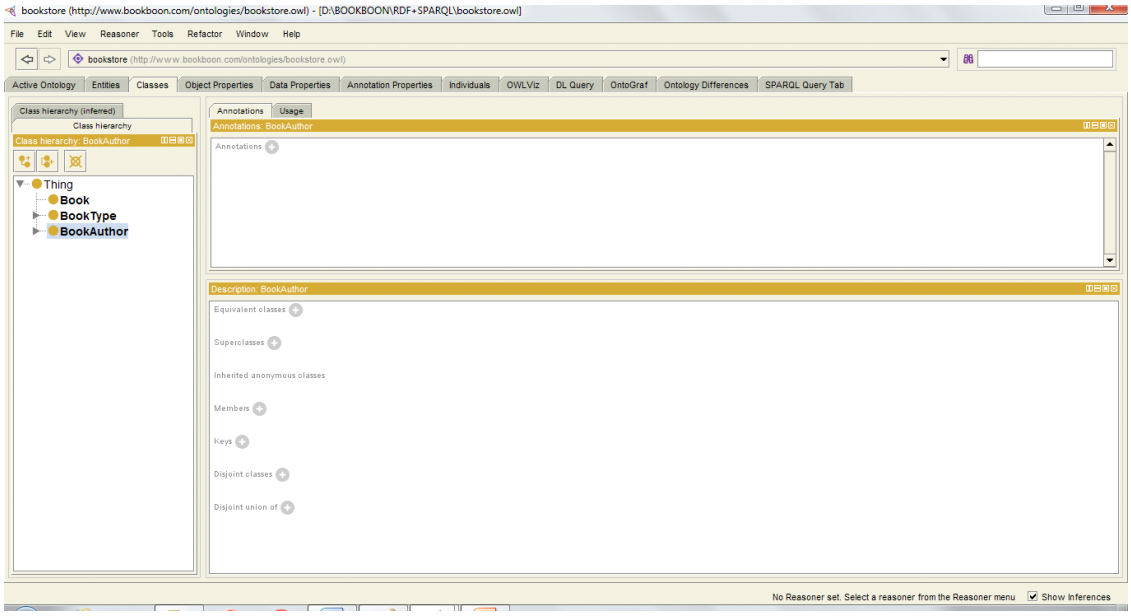
Screenshot 6.4: Protégé-classes tab

Create subclasses:

Sub-classes to be constructed are- Books, BookType and BookAuthor. They are subclasses of owl:Thing. Though there is no special naming convention, it's important that we maintain consistency. Refer to the part highlighted in the below screenshot, to know the icon used to create a sub-class.



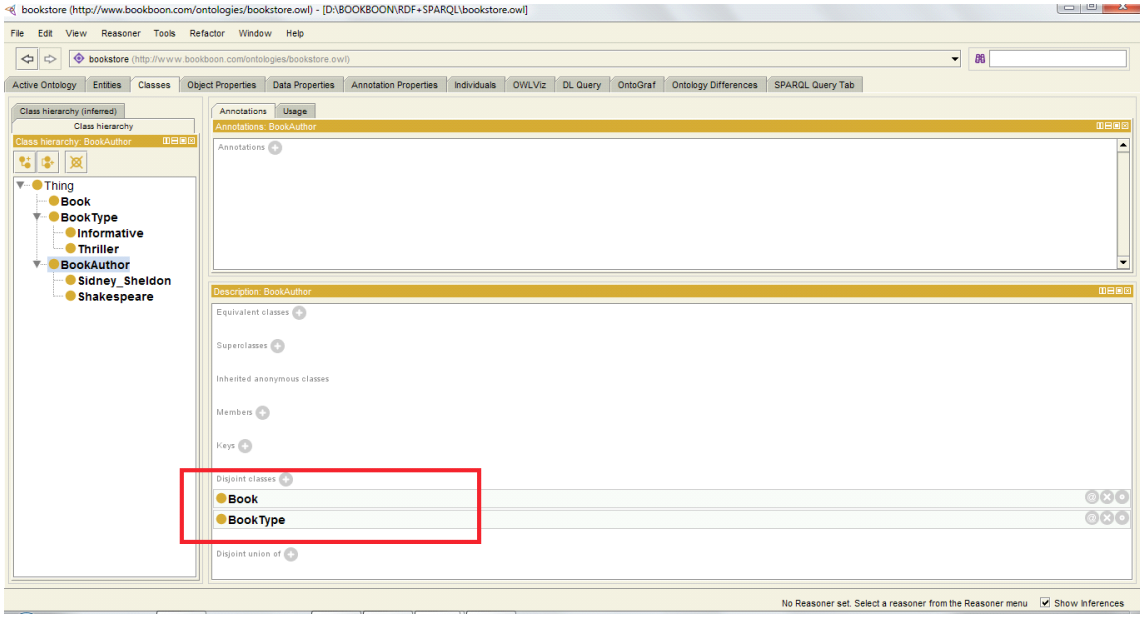
Screenshot 6.5: Creating sub-class



Screenshot 6.6: Sub-classes

Step 3:

To specify that Book, BookType and BookAuthor are disjoint classes. Add both the sub-classes to the disjoint class panel by selecting them. Refer to the below screenshot.



Screenshot 6.7: Disjoint classes

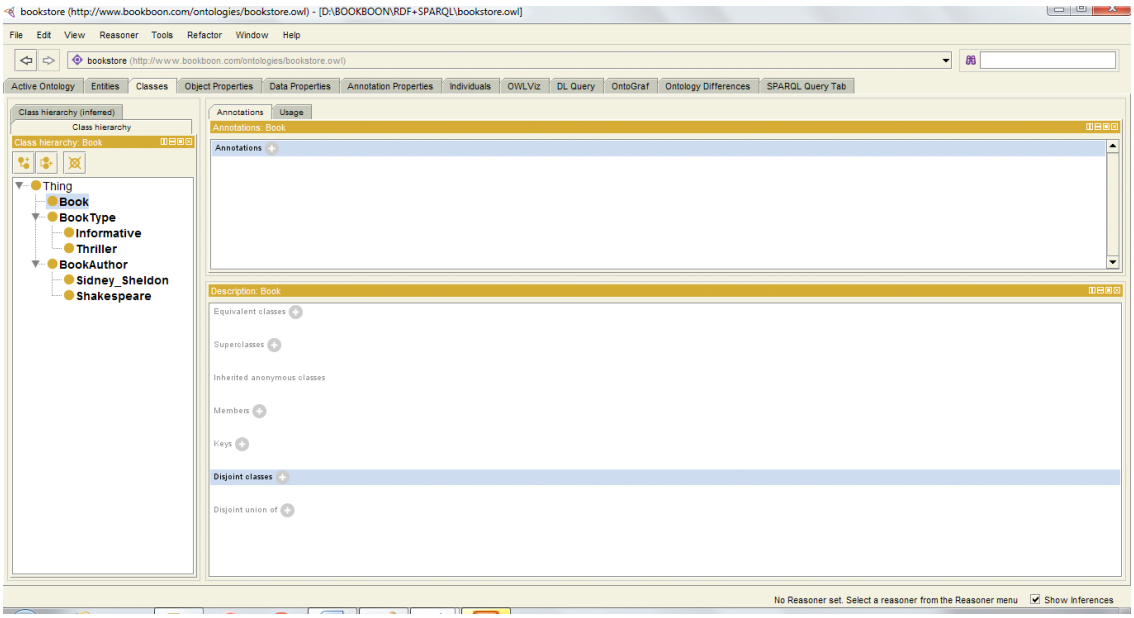
Step 4:

Create sub-classes for BookType and BookAuthor.

For example,
BookType = (Romance, Thriller)

BookAuthor = (Shakespeare, Sidney Sheldon)

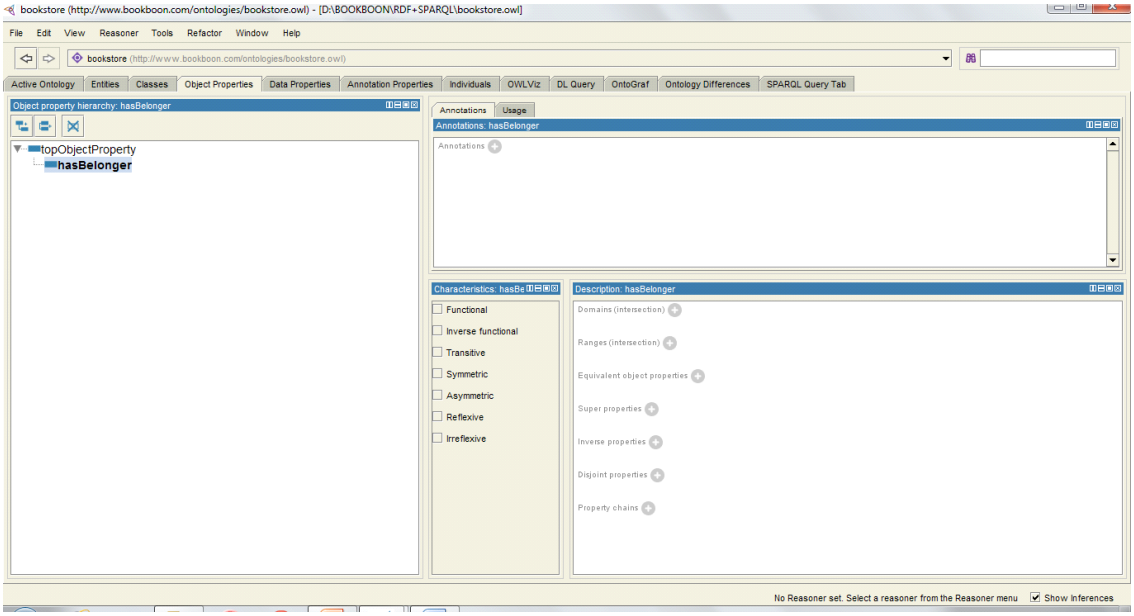
Follow the wizard to create these disjoint classes.



Screenshot 6.8 : Creating sub-classes

Step 5:

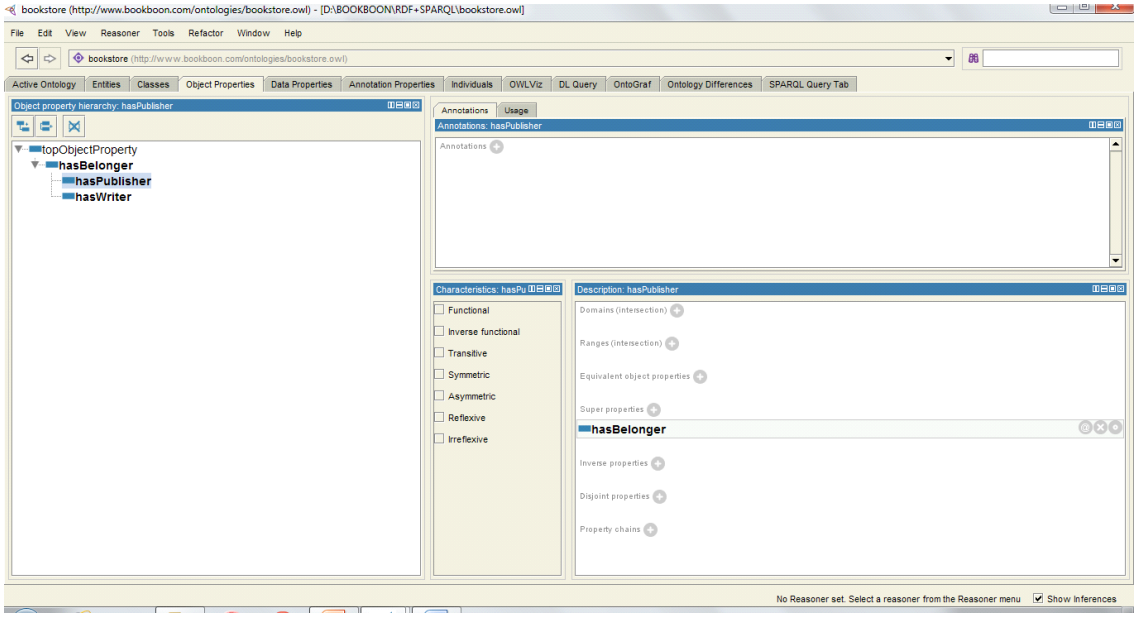
- Go to the Object properties tab.
- Click on “create Object properties”.
- Create an Object property “hasBelonger”.



Screenshot 6.8: Creating Object Properties

Step 6:

Now, select the "hasBelonger" property and add sub-properties to it. Let's create sub-properties "hasWriter" and "hasPublisher".

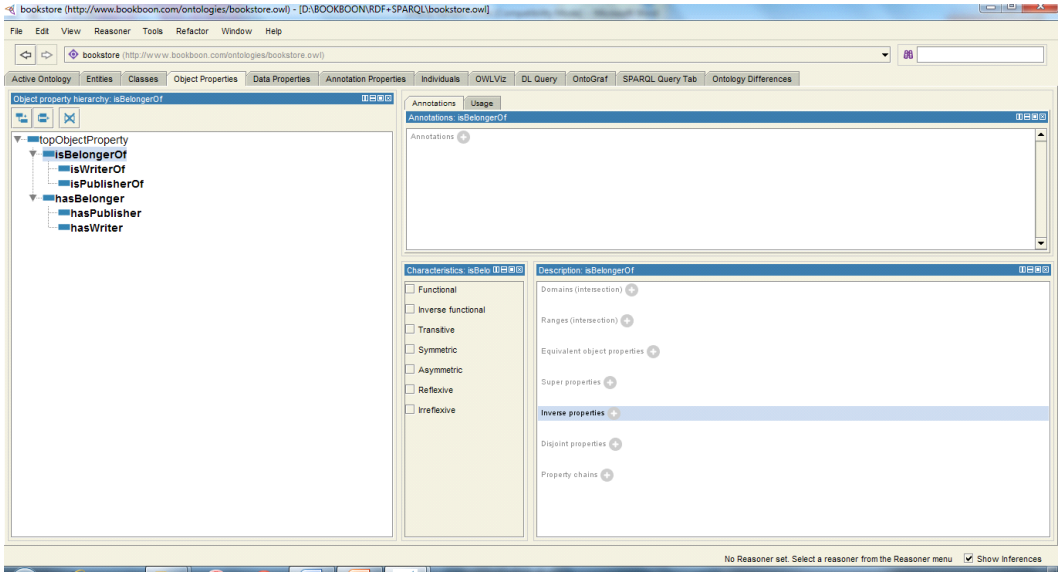


Screenshot 6.9: Creating sub-properties.

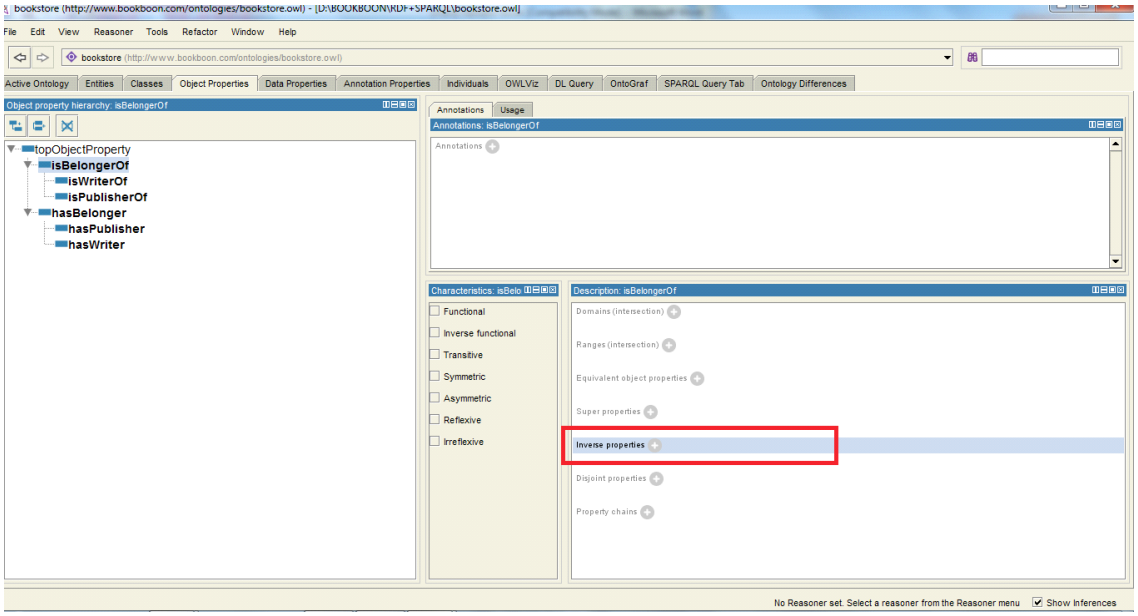
Step 7:

Create inverse properties.

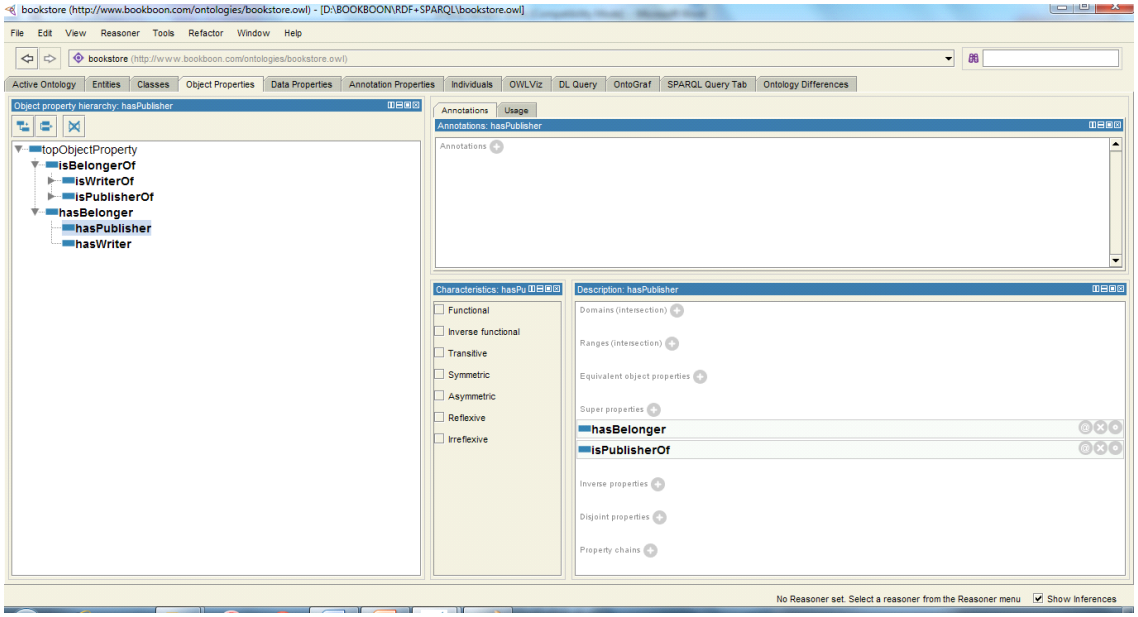
- Create a new object property called isBelongerOf.
- Press the “Set inverse property” button.
- Select “hasBelonger”.
- The inverse relation has been set up.
- Select hasPublisher
- Create the isPublisherOf as the inverse property of hasPublisher.
- Hence, isPublisherOf is the subproperty of isBelongerOf.
- Select hasWriter
- Create isWriterOf as the inverse property.
- Hence, isWriterOf is the subproperty of isBelongerOf.



Screenshot 6.10: Creating inverse properties.



Screenshot 6.11: Adding inverse properties.

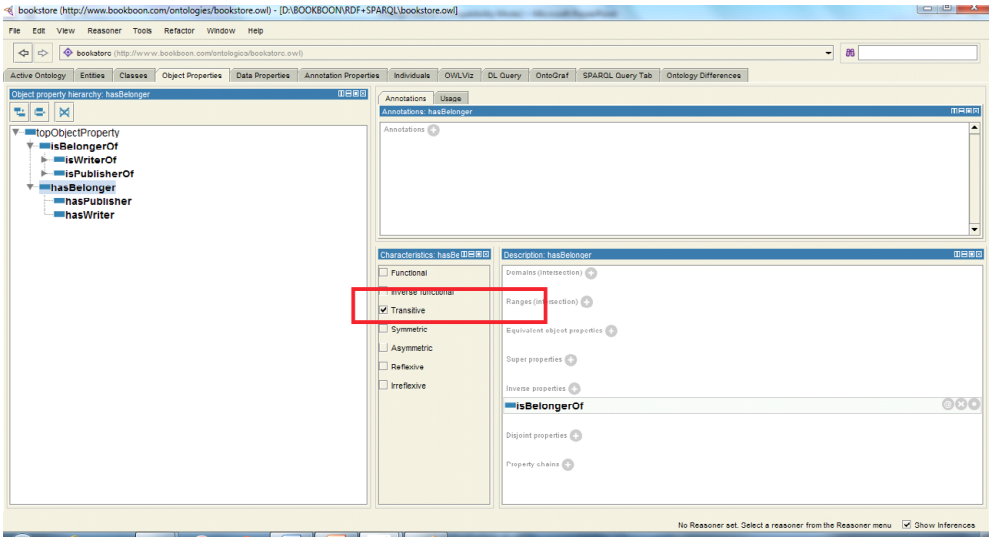


Screenshot 6.12: After completely adding inverse properties.

Step 8:

Create a transitive property.

- Select the "hasBelonger" property.
- Choose/Tick the transitive tick box.
- Select the "isBelongerOf" property, make sure that the transitive tick box is ticked.

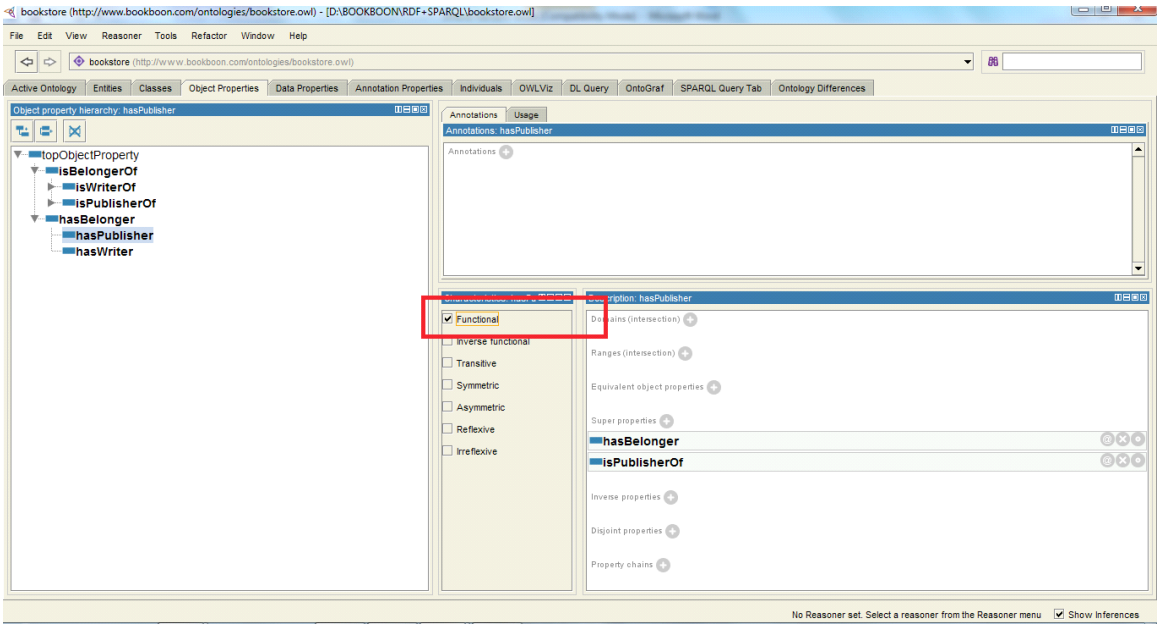


Screenshot 6.13: Creating transitive properties.

Step 9:

Create functional properties.

- Select the “hasPublisher” property
- Tick the “functional” tick box
- OWL-DL does not allow datatype properties to be transitive, symmetric or have inverse properties.

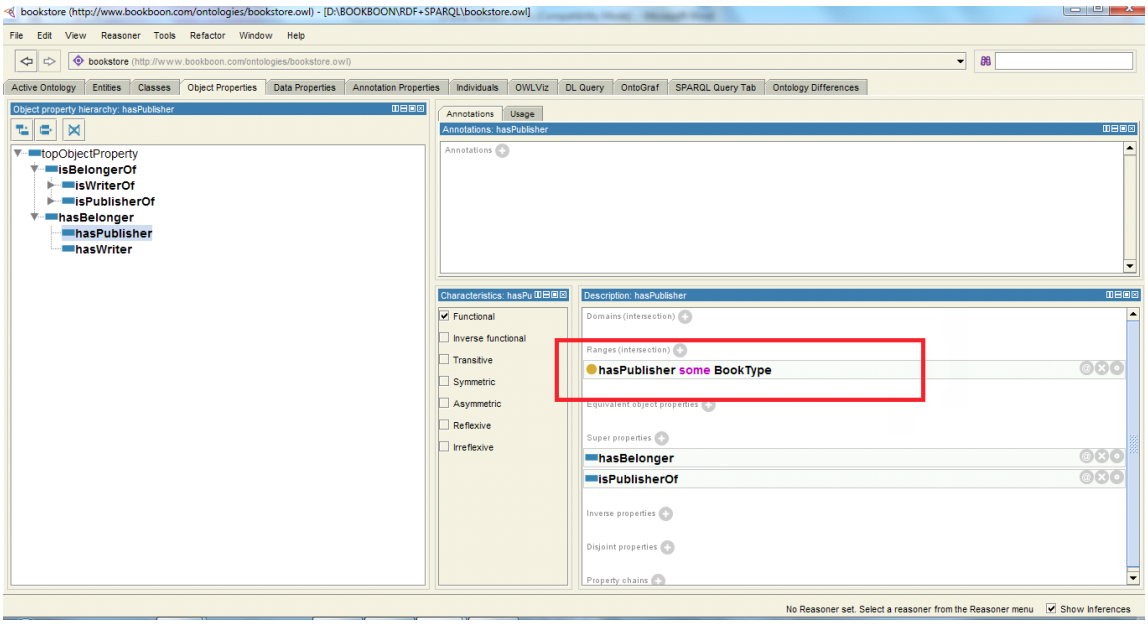


Screenshot 6.14: Creating functional properties.

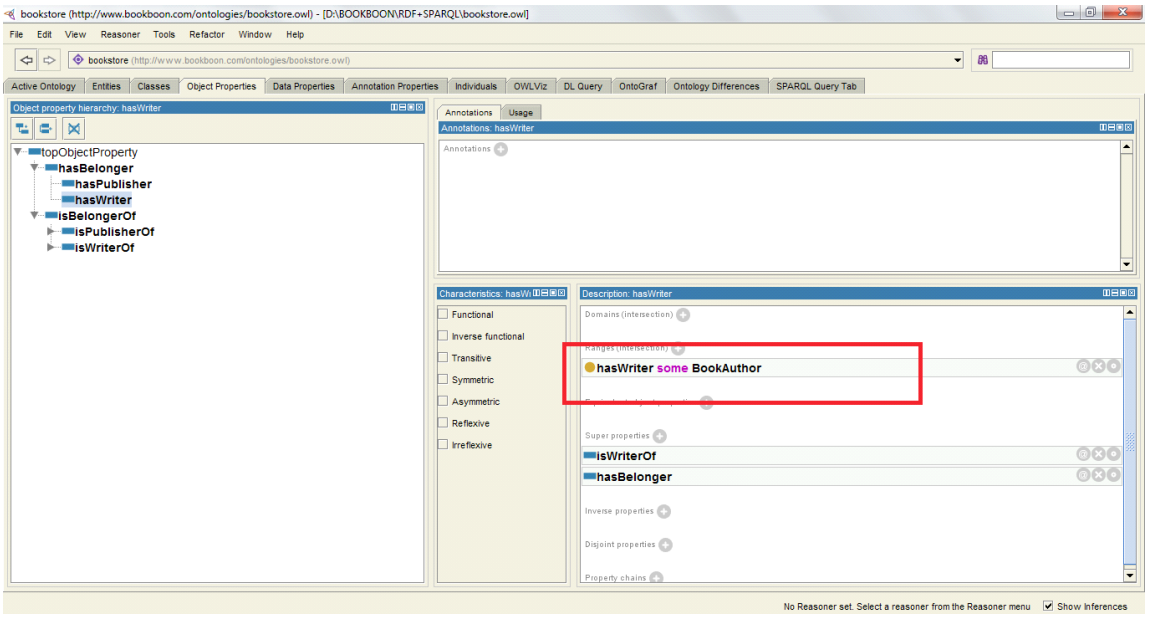
Step 10:

Specify the ranges.

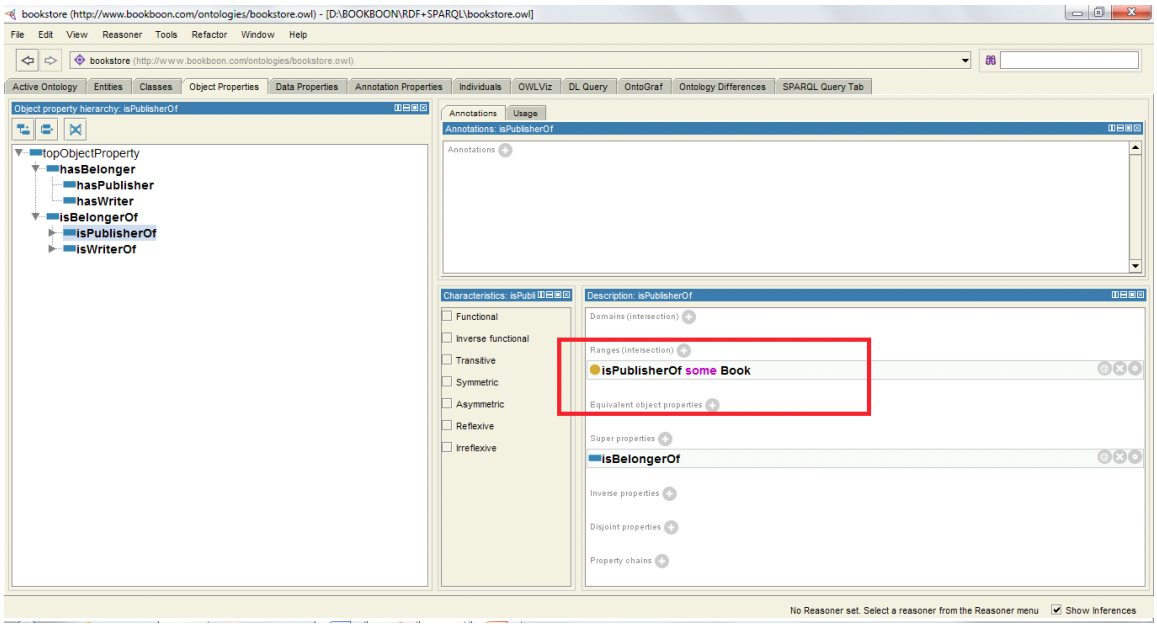
- Select hasPublisher.
- Press the range button.
- Select BookType.
- Press OK button.
- BookType must be displayed in the range list.
- When multiple classes are added to the range, they represent the union of all classes.
- Select the isPublisherOf property.
- Set the domain of the isPublisherOf property to BookType.
- Set the range of the isPublisherOf property to Book.
- The above steps will now be repeated for hasWriter.
- Select the hasWriter property.
- Specify the range as BookAuthor.
- Select the isWriterOf property.
- Specify the range as Book.



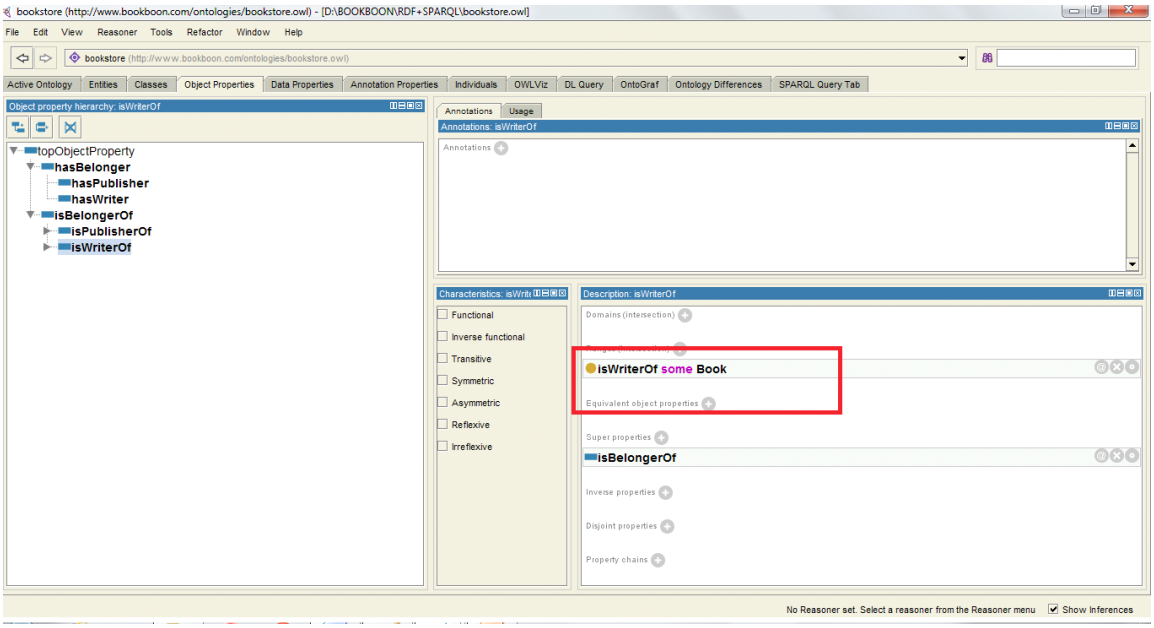
Screenshot 6.15: Specifying the ranges for hasPublisher.



Screenshot 6.16: Specifying range for hasWriter.



Screenshot 6.17: Specifying the range for isPublisherOf.

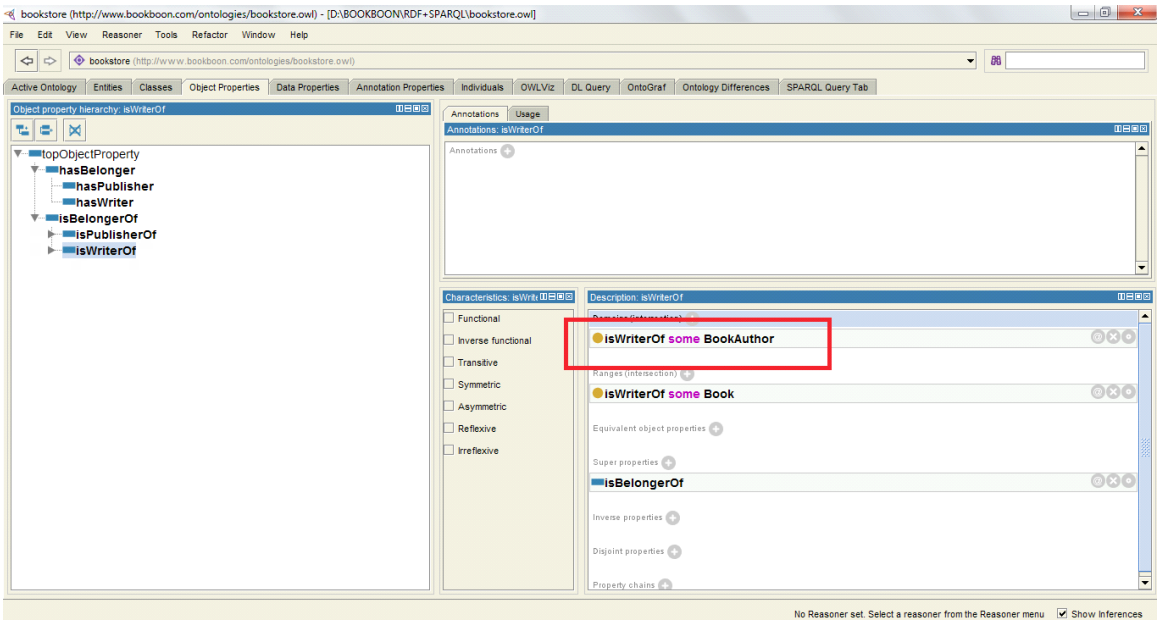


Screenshot 6.18: Specifying the range for isWriterOf.

Step 11:

Specify the domain

- Select hasPublisher property.
- Press add domain button.
- Select Book.
- Press OK.
- Book is displayed in the domain list.
- When multiple classes are added as domain, they represent the union of these classes.
- Set the domain of the isPublisherOf property to BookType.
- Select the hasWriter property.
- Specify the domain as Book.
- Select the isWriterOf property.
- Specify the domain as BookAuthor.



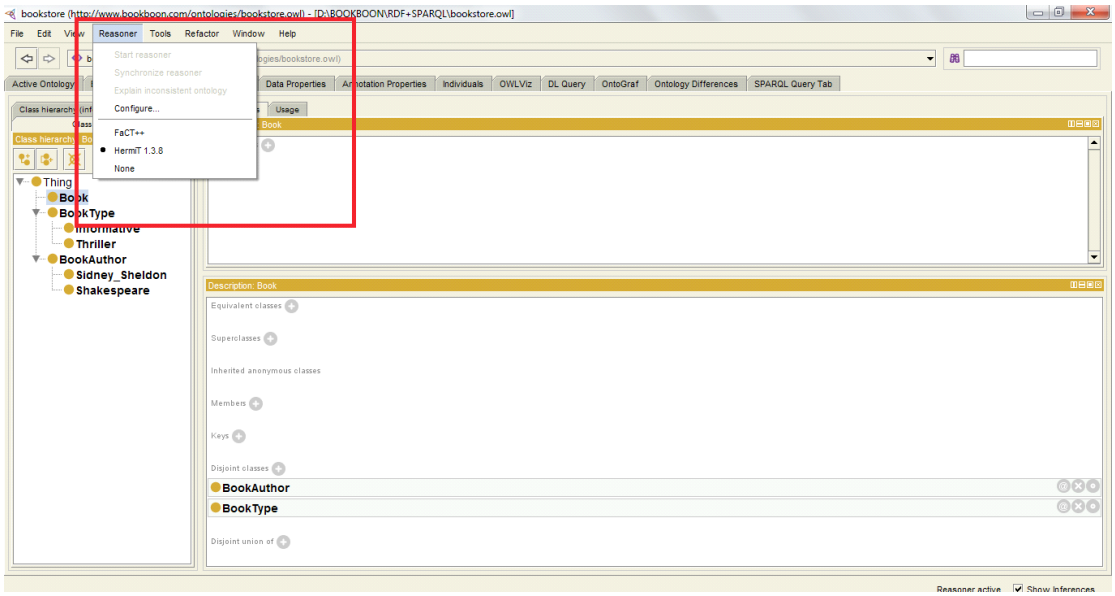
Screenshot 6.19: Specifying the domain.

Step 12:

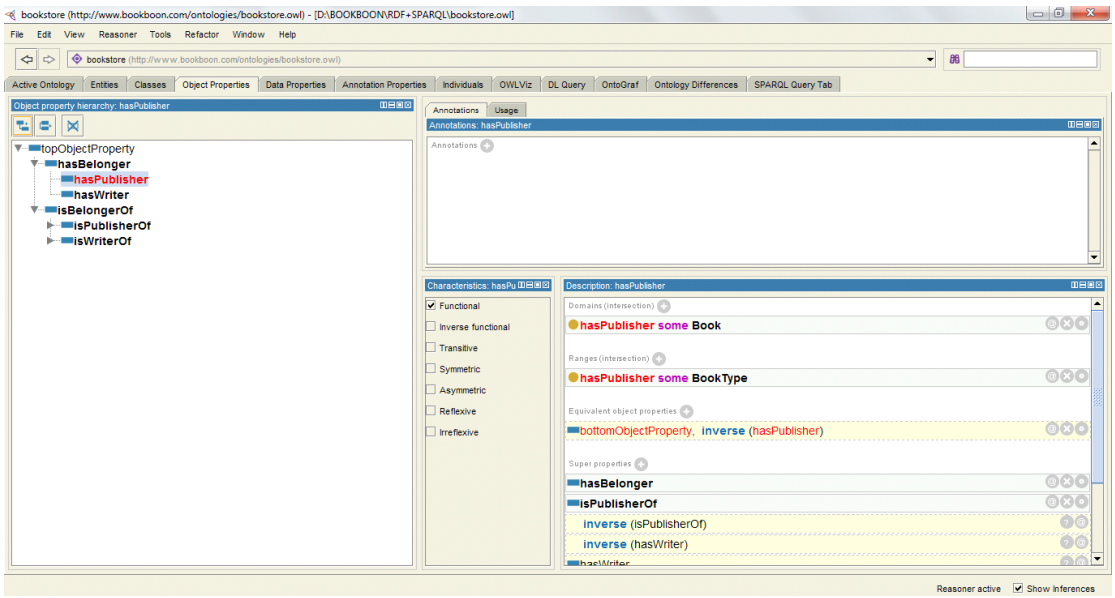
Invoke the reasoner.

Ontology described in OWL-DL can be processed by a reasoner. Go to owl —>preference, to make sure that OWL-DL is selected. The main service offered by a reasoner is to test whether or not one class is a subclass of another class. By performing such tests on all of the classes, it is possible for a reasoner to compute the inferred ontology class hierarchy.

Another reasoning service is consistency checking, i.e., to check whether or not it is possible for the class to have any instances. A class is deemed to be inconsistent if it cannot possibly have any instances.



Screenshot 6.20: Starting a reasoner.



Screenshot 6.21: Inconsistencies identified by Reasoner.

Once identified the inconsistencies are removed appropriately to ensure consistency.

7 Swoogle

Objective:

This chapter aims at exploring Swoogle which is a research level implementation of the Semantic Web.

7.1 Introduction

The Semantic Web is essentially a web-universe parallel to the web of online documents. Semantic Web Document (SWD) is well known for its semantic annotation and meaningful reference. Since no conventional search engines can take advantage of such features, a search engine customized for SWDs especially for ontologies, is needed by human users as well as software agents and services. At this stage, human users are expected to be Semantic Web researchers and developers who are interested in accessing, exploring and querying the RDF and OWL documents found on the web.

7.2 Swoogle

Swoogle is a search engine for Semantic Web ontologies, documents, terms and data published on the Web. Swoogle employs a system of crawlers to discover RDF documents and HTML documents with embedded RDF content. Swoogle reasons about these documents and their constituent parts (e.g., terms and triples) and records and indexes meaningful metadata about them in its database. Swoogle is a crawler-based indexing and retrieval system for the Semantic Web, i.e., for Web documents in RDF or OWL. It extracts metadata for each discovered document and computes relations between documents. Discovered documents are also indexed by an information retrieval system which can use either character N-Gram or URI refs as keywords to find relevant documents and to compute the similarity among a set of documents. One of the interesting properties we compute is rank, a measure of the importance of a Semantic Web document.

Screenshot 7.1: User Interface of Swoogle.

7.3 History of Swoogle

Swoogle is a research project being carried out by the Ebiquity research group of the Computer Science and Electrical Engineering Department at the University of Maryland, Baltimore County (UMBC). Partial research support was provided by DARPA contract F30602-00-0591 and by NSF, by awards NSF-ITR-IIS-0326460 and NSF-ITR-IDM-0219649. Contributors include Tim Finin, Li Ding, Rong Pan, Anupam Joshi, Pavan Reddivari, Joel Sachs, Pranam Kolari, Akshay Java, Lushan Han, Yun Peng, R. Scott Cost, Sandor Dornbush and Vishal Doshi. Swoogle indexes only Semantic Web documents, currently including those written in RDF/XML, N-Triples, N3(RDF) and some documents that embed RDF/XML fragments. The data presented at Swoogle are collected from the public accessible World Wide Web. Swoogle has a privacy policy in English and a crawling policy.

7.4 Implementation of Swoogle

Swoogle adopts a hybrid approach to harvest the Semantic Web, including manual submission, Google-based meta-crawling, bounded HTML crawling and RDF crawling. Swoogle's statistic page is under re-construction now. Swoogle has over 1.4 million Semantic Web documents and 290 million triples indexed. Swoogle is mainly written in Java (JDK 1.4.2), and its web services are provided through Apache Tomcat server. The front-end of the Swoogle website is written in PHP. Currently, Swoogle stores its data in a MySQL database (mysql 4.1.16). The entire system runs on a Linux platform (Fedor Core 4). Swoogle is designed as a repository of URLs but not as a triple store, so it does not store all triples encountered. However, applications may be built on top of Swoogle that index a specific class of Semantic Web data and provide inference support using triple stores.

Swoogle tries to enhance the following features of the Semantic Web:

Finding appropriate ontologies:

Failing to find a proper ontology always leads to the creation of a new ontology, which is often customized to be reused. Swoogle helps users to find ontologies containing specified terms, and users may even qualify the type (class or property) of a term. Moreover, the ranking mechanism sorts ontologies by their popularity. This feature is believed to not only ease the burden of marking up data but also contribute to the emergence of canonical ontologies.

Finding instance data:

In order to help users integrate Semantic Web data distributed on the Web, Swoogle enables querying SWDs with constraints on the classes and properties used by them.

Characterizing the Semantic Web:

By collecting meta-data, especially inter-document relations, about the Semantic Web, Swoogle reveals interesting structural properties such as 'how the Semantic Web is connected?', 'how ontologies are referenced?', and 'how an ontology is modified externally?'.

8 Case-Study Related To Semantic Web

Objective:

This chapter consists of a case-study which can be converted into a full-fledged Semantic Web project. Students willing to take-up research projects on Semantic Web and studying this book with an aim to build a project can work on the case study.

8.1 An introduction to E-Commerce

E-Business

E-business is the application of Information and Communication Technologies (ICT) in support of all the activities of the business. E-business will be a secure, flexible and integrated approach combining the systems and processes that run core business operations. It is the process of using Web technology to help the streamline processes, improve productivity, increase efficiencies and enable organizations to easily communicate with partners, vendors and customers, connect back-end data systems and transact commerce in a secure manner.

E-Commerce

Electronic Commerce (EC) or simply e-commerce is an emerging concept that describes the buying and selling of products, services and information via computer networks, including the Internet.

8.2 Challenges to E-Commerce

Peer to peer approach plays a major role in developing E-Commerce to its fullest potential.
Following are some of the challenges faced by E-Commerce and Semantic Web solution:

- Mechanized support is needed in finding and comparing vendors and their offers.
  Currently, almost all of this work is done manually which seriously hampers the scalability of electronic commerce. Semantic Web technology can make it machine-processable.

- Mechanized support is needed in dealing with numerous and heterogeneous data formats. Various 'standards' exist on how to describe products and services, product catalogues and business documents. Ontology technology is required to define such standards better. Efficient bridges between different terminologies are essential for openness and scalability.

- Mechanized support is needed in dealing with numerous and heterogeneous business logics.

8.3 Current scenario

E-commerce business model

A Model is the systematic or structured approach which instructs the ways to be followed in order to perform a business activity. The business model spells out how a company plans to make money and how it is competitively positioned in an industry.

- A business model is a method of doing business by which a company can generate revenue to sustain itself. Organizations must define and execute a strategy to be successful in e-commerce.
- A business model that aims to use and leverage the unique qualities of the Internet and the World Wide Web is called E-Business Model.
- The components that are contained within a business model address all functions of a business, including factors such as the expenses, revenues, operating strategies, corporate structure, and sales and marketing procedures. (A company's policy, operations, technology and ideology define its business model)

E-commerce business model:

A business model that aims to use and leverage the unique qualities of the Internet and the World Wide Web is called an E-commerce business model.

Key ingredients of a business model

- Value Proposition.
- Revenue model.
- Market opportunity.
- Competitive environment.
- Competitive advantage.
- Market strategy.
- Organizational development.
- Management team.

1. Value proposition

Value proposition defines how a company's product or service fulfills the needs of customers. It aims to answer the following questions –

- Why will customers choose to do business with your firm instead of another company?
- What will your firm provide that other firms do not and cannot?

2. Revenue model

Revenue model describes how the firm will earn revenue, produce profits, and produce a superior return on invested capital.

Following are the well-known e-commerce revenue models:

- Advertising model
- Subscription model
- Transaction fee model
- Sales model
- Affiliate model

- Advertising revenue model
  In advertising model a company provides a forum for advertisements and receives fees from advertisers.
  Example: yahoo

- Subscription revenue model
  In subscription model, a company offers its users, content or services and charges a subscription fee for access to some or all of its offerings.
  Example: Wall Street Journal

- Transaction fee revenue model
  In transaction fee model, a company receives a fee for enabling or executing a transaction.
  Example: eBay

- Sales revenue model
  In a sales revenue model, a company derives revenue by selling goods, information, or services.
  Example: Amazon

- Affiliate revenue model
  In an affiliate revenue model, a company steers business to an affiliate and receives a referral fee or percentage of the revenue from any resulting sales.
  Example: MyPoints

3. Market opportunity

- Market opportunity refers to the company's intended market space and the overall potential financial opportunities available to the firm in that market space.
- It is defined by the revenue potential in each of the market niches where you hope to compete.

Market space

Market space is the area of actual or potential commercial value in which a company intends to operate.

4. Competitive environment

Competitive environment refers to the other companies operating in the same marketplace selling similar products.

It is impacted by the following factors:

- Number of competitors who are active
- How large are their operations?
- The market share of each competitor
- How profitable these firms are?
- How they price their products?

5. Competitive advantage

- Competitive advantage is achieved by a firm when it can produce a superior product and/or bring the product to market at a lower price than most, or all, of its competitors
- It is achieved because a firm has been able to obtain differential access to the factors of production that are denied by their competitors.

6. Market strategy

- The plan you put together that details exactly how you intend to enter a new market and attract new customers is the market strategy.
- Best business concepts will fail if not properly marketed to potential customers

7. Organizational development

- It describes how the company will organize the work that needs to be accomplished.
- Work is typically divided into functional departments.
- Move from generalists to specialists as the company grows.

8. Management team

- Employees of the company, responsible for making the business model work, comprise of the management team.
- Strong management team gives instant credibility to outside investors. A strong management team may not be able to salvage a weak business model
- The management team should be able to change the model and redefine the business whenever needed.

8.3.1 Major E-business model

- Business-to-Consumer (B2C)
- Business-to-Business (B2B)

Business to consumer business model

- B2C (or Extranets) is just web-enabled relationships between existing partners. They are run by a single company seeking to lower the cost of doing business with its current suppliers or individual customers.

Examples:

Amazon.com, Egghead.com

Business to Business

- 'B2B' is business-to- business commerce conducted over the Internet (called B2B e-commerce space, or e-marketplaces)

B2B applications:

- Advertising
- Auctioning
- Procurement
- Channel management
- E-commerce

8.3.2 Unique features of E-commerce technology

1. **Ubiquity**
   E-commerce alters industry structure by creating new marketing channels and expanding size of overall market. It creates new efficiencies in industry operations and lowers the cost of firms' sales operations. It also enables new differentiation strategies.

2. **Global Reach**
   It changes industry structure by lowering barriers to entry, but greatly expands market at the same time. It lowers the cost of industry and firm operations through production and sales efficiencies. It also enables competition on a global scale.

3. **Universal Standards**
   It changes industry structure by lowering barriers to entry and intensifying competition within an industry. Lowers costs of industry and firm operations by lowering computing and communication costs and enables broad-scope strategies.

4. **Richness**
   It alters industry structure by reducing strength of powerful distribution channels. It changes the industry and firm operations costs by lessening reliance on sales force and enhances post-sale support strategies.

5. **Interactivity**
   It alters industry structure by reducing the threat of substitutes through enhanced customization. Reduces industry and firm costs by lessening reliance on sales force and enable differentiation strategies.

6. **Personalization/Customization**
   It alters industry structure by reducing threats of substitutes, raising barriers to entry and reduces value chain costs in industry and firm by lessening reliance on sales forces.

7. **Information Density**
   It changes industry structure by weakening powerful sales channels, shifting bargaining power to consumer and reduces industry and firm operations costs by lowering costs of obtaining, processing, and distributing information about suppliers and consumers.

8.4 Case study 1 – Implementing a Virtual Travel Agency in Semantic Web

Aim:

To implement a Virtual Travel Agency in Semantic Web. Virtual Travel Agency (VTA) acts as an intermediary service between the customers and tourism service providers. The tourism service providers, here, will be commercial companies that provide flight, hotel, car-rental services etc. The output will provide tourism package to customers by aggregating services of various tourism service providers.

Description of the project scenario:

Imagine a VTA which is an end-user service, providing e-Tourism services to customers. These services can cover all kinds of information services concerned with tourism. For example, information about events and sights in an area, services that support booking of flights, hotels, rental cars, etc. Such VTAs exist even now in the market. Currently, those portals are accessible via html sites. The partners of the VTA are integrated via conventional B2B integration.

By applying Semantic Web Services, a VTA will be easily able to configure and invoke Web Services provided by several e-Tourism suppliers and aggregate them into new customer services. Such VTAs providing automated e-Tourism services to end users via different interfaces and can be more easily reconfigured according to the actual needs.

Our VTA aggregates Web Services of different tourism service providers. In a nutshell, it provides the following functionality:

A customer uses the VTA service as the entry point for his requests. The end-user services are aggregated by the VTA by invoking and combining Web Services offered by several tourism service providers. To facilitate this, there can be a so called "umbrella" contract between the service providers and the VTA for regulating usage and allowance of the Web Services.

8.4.1 Drawback of the current system

- Current VTA's maintain the data in their own database. The offers that we get are just from the fixed set of services that they acquire by collaborating with different service providers.
- The service providers do not directly come in contact with the customers with their own set of offers. This is because of the variation in the format of the database used by these service providers. For example the field name for "product identification number" may be "prod_id" in service provider-1's database and "product_id" in service providers-2's database.
- Hence it becomes difficult to integrate these services together and provide a consolidated package to the customer.

8.4.2 What the target system does?

- The target system will retrieve the services from the database of the service provider itself instead of retrieving it from the database maintained by the VTA.
- This is done by Ontology mapping which maps the field names with their appropriate meaning and different possible field names.

VTA's services:

A Web Service of a VTA offers end-user services for searching and booking hotel and flight tickets. This Web Service is composed of other Web Services, namely accommodation and transportation services that are provided and published by different companies and are registered with the VTA.

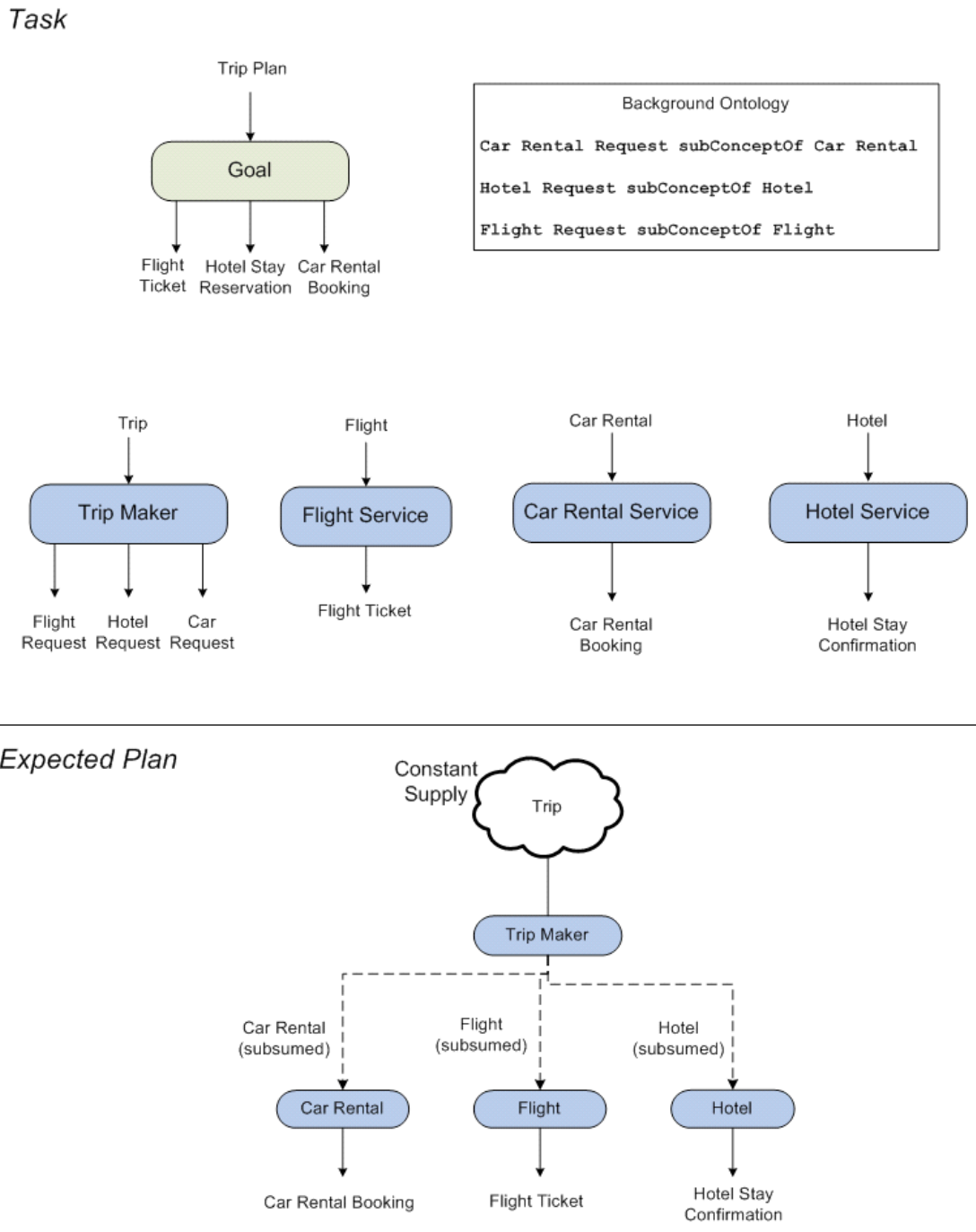


Figure 8.1: Expected output of VTA.

8.5 Architecture of the proposed system

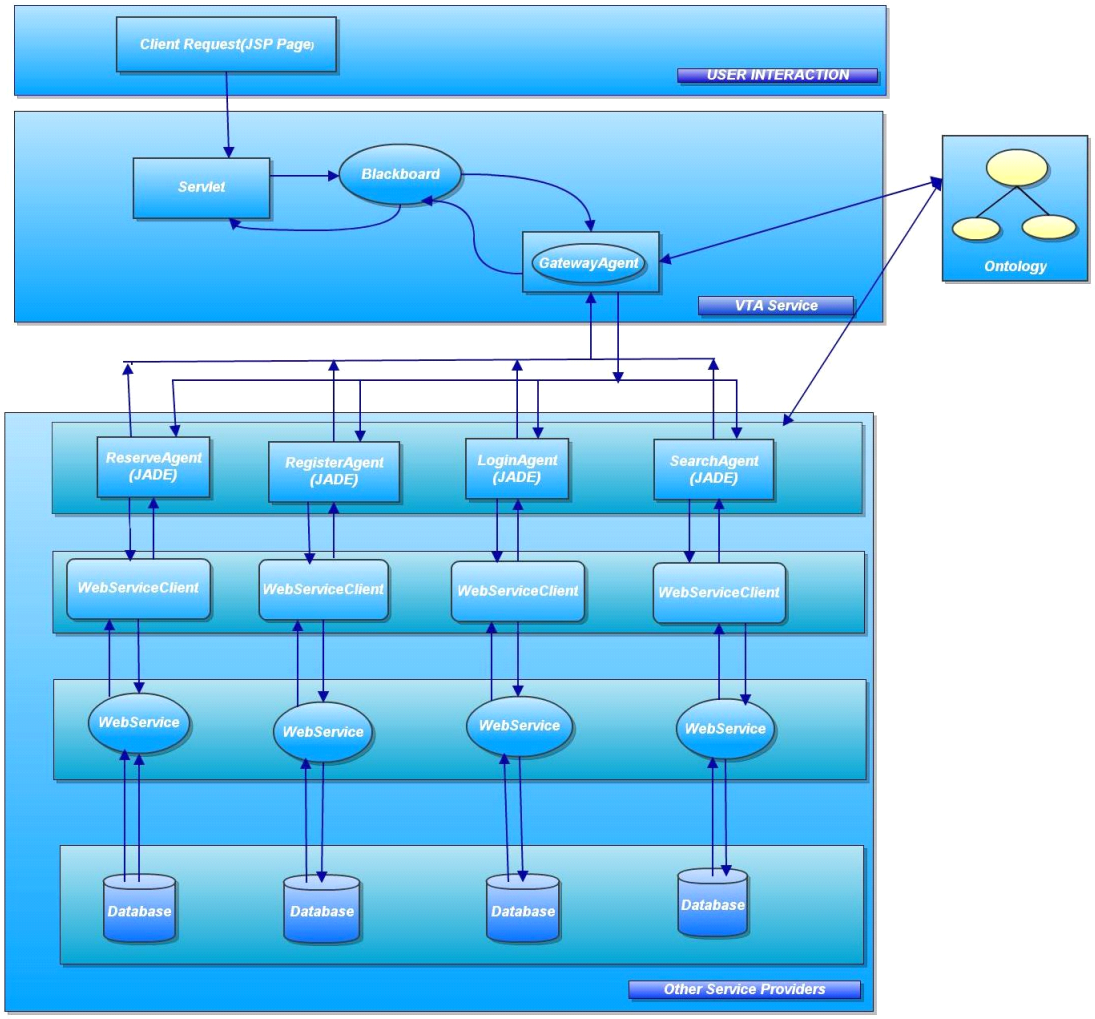


Figure 8.2: VTA Architecture.

Technologies to implement VTA:

- JADE
- HTML 5, CSS, Java script
- Servlets

JADE:

JADE (Java Agent Development Environment) is a middleware that facilitates the development of multi-agent systems. It includes

- A runtime environment where JADE agents can "live" and that must be active on a given host before one or more agents can be executed on that host.
- A library of classes that programmers have to/can use (directly or by specializing them) to develop their agents.
- A suite of graphical tools that allows administrating and monitoring the activity of the running agents.

Each running instance of the JADE runtime environment is called a Container as it can contain several agents. The set of active containers is called a Platform. When an agent A communicates with another agent B, a certain amount of information I is transferred from A to B by means of an ACL message. Inside the ACL message, I is represented as a content expression consistent with a proper content language and encoded in a proper format. Both A and B have their own way of internally representing I. Hence it's clear that each time agent A sends a piece of information I to agent B, A needs to convert his internal representation of I into the corresponding ACL content expression representation and B needs to perform the opposite conversion. Moreover B should also perform a number of semantic checks to verify that I is a meaningful piece of information. The support for content languages and ontologies provided by JADE is designed to automatically perform all the above conversion and check operations

Process:

In the browser the user creates an event and it generates a POST message. The servlet handles it and the "sendmessage" action is invoked. The action creates a new BlackBoard object which will be the message channel between the GatewayAgent and the servlet. The GatewayAgent gets the dashboard object created previously and extracts the recipient and the message. After that, it sends the message. ServerAgent who is now the recipient, responds to the GatewayAgent. ServerAgents referring to the ontological description invoke appropriate client web service that in turn invokes the web service on the service providers' domain that retrieves result from its database and passes it on. When it reaches the GatewayAgent, the GatewayAgent packs the reply and sends it via BlackBoard to the servlet. The servlet forwards it to the browser.

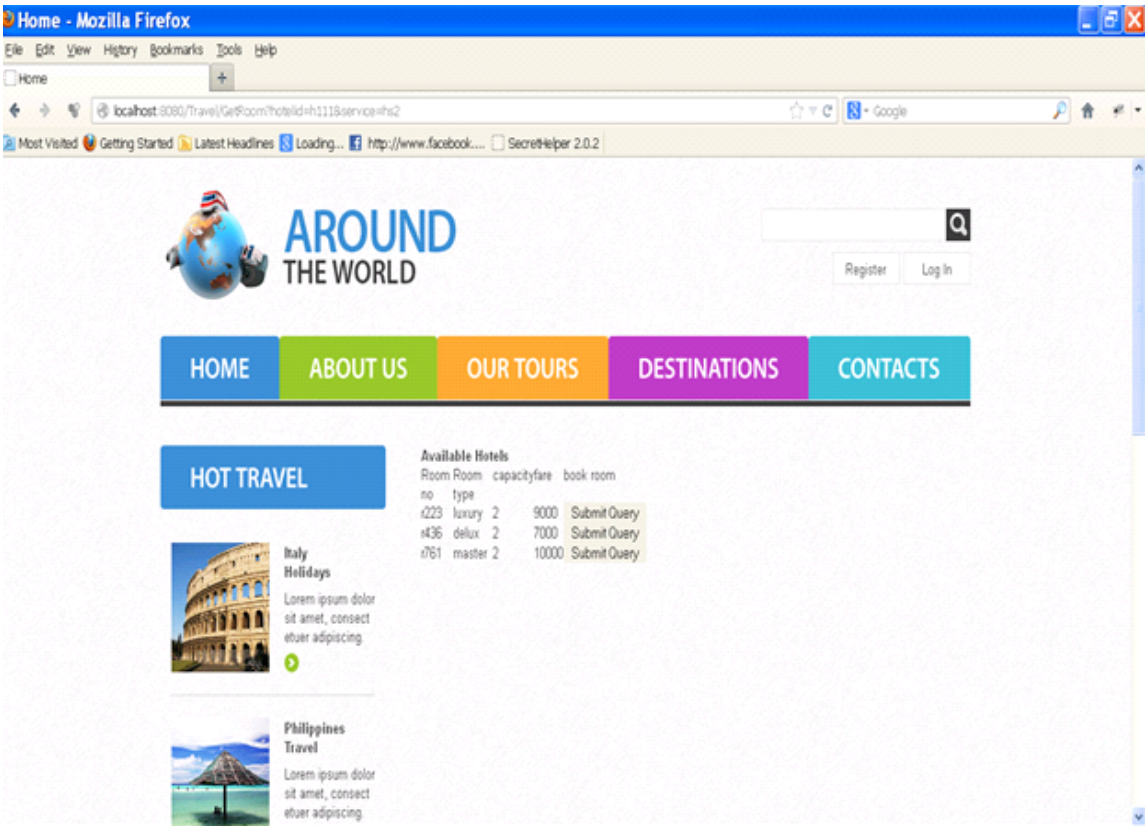


Figure 8.3: Sample output of VTA.

Conclusion:

Thus, we can conclude that this budding technology has a lot of scope. The only problem it faces is that all the tools and interfaces developed to implement services in Semantic Web, such as WSMT (Web Service Modeling Toolkit), are all at their rudimentary stage of development unlike tools in Web 2.0 such as Netbeans which are already well-developed and ready to use. Hence the conclusion is that, if we are able to concentrate more on developing these tools into a full-fledged interface, this technology of Semantic Web can come out with wonderful inventions.

References:

1. www.wikipedia.org
2. A Semantic Web Search and Metadata Engine *
   (Li Ding, Tim Finin, Anupam Joshi, Yun Peng, R. Scott Cost, Joel Sachs, Rong Pan, Pavan Reddivari, Vishal Doshi – Department of Computer Science and Electronic Engineering, University of Maryland Baltimore County, Baltimore MD 21250, USA)
3. Virtual Travel Agency: A Multi-Agent-System that implements Semantic Web Services. (Swati Ringe, Dhana Nandini, Gaurav Sharma, Mayuri Rege – Computer Engineering Department, University of Mumbai, M.G. Road, Fort, Mumbai-400 032. India)
4. Semantics and Complexity of SPARQL
   (Jorge Pérez1, Marcelo Arenas2, and Claudio Gutierrez – Universidad de Talca, Chile, Pontificia Universidad Cat'olica de Chile, Universidad de Chile)
5. Web to Semantic Web & Role of Ontology
   (Zeeshan Ahmed, Detlef Gerhard Mechanical Engineering Informatics and Virtual Product Development Division, Vienna University of Technology, Getreidemarkt 9/307 1060 Vienna, Austria)
6. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications
   (Holger Knublauch, Ray W. Fergerson, Natalya F. Noy and Mark A. Musen Stanford Medical Informatics, Stanford School of Medicine 251 Campus Drive, Stanford, CA)
7. http://swoogle.umbc.edu/
8. http://obitko.com/tutorials/ontologies-semantic-web/
9. www.linkeddatatools.com